Proceedings of the 2nd National Conference; INDIACom – 2008
Computing For Nation Development, February 08 – 09, 2008
Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

The Design of a Pedagogical Operating System

Pinaki Chakraborty
School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi – 110067
E-mail: pinaki_chakraborty_163@yahoo.com

R. G. Gupta
Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi – 110067
E-mail: rgguptajnu@yahoo.co.in

ABSTRACT
Operating systems are indispensable part of all computer systems used today. The concepts and principles used to design and implement operating systems have been evolving over the decades. However, the researchers are still busy in improving the operating systems and imparting new properties in them. One such desired property is that of being pedagogical in nature. In the current paper, the primal design of a pedagogical operating system is being developed. To attain respectable modularity and flexibility, the multiple server microkernel based architecture has been used. Additionally, object oriented methodologies and software engineering principles has been employed to obtain a well designed and well documented operating system. The pedagogical operating system whose design is being developed in this paper is expected to be helpful in teaching and learning courses on operating systems in various universities and other educational organizations on its completion. Furthermore, the proficient design of the operating system, obtained by employing various august paradigms and introducing new features, aims at enlivening academic atmosphere and promoting scientific innovation.

KEYWORDS
Operating system, pedagogical operating system, microkernel, object oriented operating system.

INTRODUCTION

The design stage of the development of any software is important as it comprehensively defines the structure as well as the behavior of the proposed software. Consequently, all the subsequent stages of the development process of the software depend heavily on this stage. The design stage transfers the requirement specification to a structure suitable for ready implementation. For a large and complex piece of software, like an operating system, it is customary to first develop a primal design and then to explicate it to obtain the detailed design.

The current paper principally discusses the design stage of the development of a pedagogical operating system to the point of the genesis of its primal design. The paper begins with a discussion on the fundamental design paradigms being used in the development process. Then the paper presents the primal design of the operating system and finally provides brief overviews of the two novel features being employed in this operating system.

THE DESIGN PARADIGMS

The operating system in discussion is being developed following a few well established paradigms of Computer Science. The operating system is being developed to serve as a pedagogical tool. It is conceptualized to have a multiple server microkernel based architecture. Moreover, the operating system is being developed using object oriented methodologies and the development process is being guided by the principles of software engineering. Each of these paradigms serves a particular purpose and the justifications for selecting them are discussed in the following five subsections.

The Pedagogical Objective

Students learn by experimenting and not merely by listening [1]. This opinion was originally made in the context of courses on operating systems. For better understandings of the students, there should be some experiments to study the structure and the behavior of the operating systems. In other disciplines of Computer Science, pedagogical software tools have been developed for elucidating recalcitrant concepts [2]. Similarly, if a well documented pedagogical operating system, along with its complete source code, is available to the students then they can study the structure as well as the working of the system. Therefore, the need of the hour is some well documented pedagogical operating systems.

The Microkernel Based Architecture

The operating systems have always been comparatively large and complex programs. In an early attempt to simplify their structures, Hansen [3] put forward the concept of 'operating system nucleus'. This concept has evolved over the years and is now known as 'microkernel'. A microkernel is a minimal operating system kernel which, in its purest form, offers no operating system services but provides necessary mechanisms to implement the various operating system services. Although there is no consensus regarding the services that should be provided by a microkernel, most microkernels typically provide

minimal process and memory management and interprocess communication [4]. The microkernel is also the only part of the operating system that executes in the privileged kernel mode. The operating system services not furnished by the microkernel are implemented by a number of modules arranged in one or more layers above the microkernel layer. The microkernel based operating systems are better maintainable and modifiable, and can be easily ported to different hardware platforms. Moreover, the microkernel based architecture provides better reliability and security [4]. Although the microkernel based operating systems have been widely criticized for alleged performance problems, recent studies have shown that the microkernel based architecture can essentially provide competitive performance [5].

The Multiple Server Design

Among all the microkernel based architectures, the multiple server microkernel based operating system architecture is most promising due to its superior modularity [6]. In a multiple server microkernel based operating system, the various system components communicate with each other by passing messages. A major advantage of this approach is that the source code of microkernel remains small in size and consequently it is easy to keep it error free [7]. The faults in the other components of the operating system can create only localized problems. But in no case these faults can bring down the entire system. As a demerit of this design, there is a performance penalty because message passing implies an overhead of building and copying of messages. However, it has been observed that the message passing overhead is negligible in a well designed multiple server microkernel based operating system and it hardly affects the performance of the system.

The Object Oriented Approach

An operating system is said to be an object oriented operating system if and only if it internally uses an object model. Following an object model, an operating system can be designed and implemented as a set of types, each of which can be thought of as a kind of resource [8]. Some of these resources, like the input/output devices, have direct physical realizations at the computer hardware level. Alternatively, some resources may be abstract and without any realization at the computer hardware level. The concepts of files, streams, mailboxes and device drivers are good examples of object orientation. Actually, all of them are abstract data types. They have various methods in the form of system calls whose behavior varies depending on the type of the object actually used to invoke them. The implementation details of these abstract data types are hidden from the rest of the system and they typically use standard object oriented techniques, like inheritance, polymorphism and delegation, in their underlying codes.

The Software Engineering Based Guidance

The development process is being guided by software engineering principles to have an efficient development process and a timely delivery of the operating system. The Waterfall Model is being followed to develop the operating system. The justification for selecting this particular software lifecycle model is that the requirements are known in advance and the development process is supposed to be protracted but straightforward.

All necessary documents are being written regularly. The software requirement specification has been written and the design stage has commenced on its basis. Currently, the software design document is being written and it will be complete after the detailed design is completed. Since an operating system is not a data processing software, entity relationship diagrams and dataflow diagrams have little meaning in its design. However, for an operating system categorizing its probable users into two or more categories the use case diagram may be helpful in illustrating the design. Furthermore, there are some documents and diagrams particularly important for the development of operating systems, like the component malfunctioning policy scripts and process lifecycle diagram, which are being developed. An installation guide and a user manual will be also written for the operating systems at a later stage.

THE PRIMAL DESIGN

The operating system being developed have a multiple server microkernel based architecture with three distinct layers of system components between the microkernel and the application software layer. In other words, the system consists of five layers in all (Fig. 1). Layer 1 is the lowest layer and it is closest to the computer hardware. It contains the microkernel and the clock driver. The microkernel and the clock driver are the only parts of the operating system executed in the privileged kernel mode. Layer 2 contains the device drivers for all the important input/output devices. In a contrast to most operating systems, these device drivers are executed in the unprivileged mode. Layer 3 contains system components for consolidation and optimization of the system services provided by the layers below it. Layer 4 contains several servers each providing a particular type of service. Finally, Layer 5 contains the user programs and is the topmost layer.

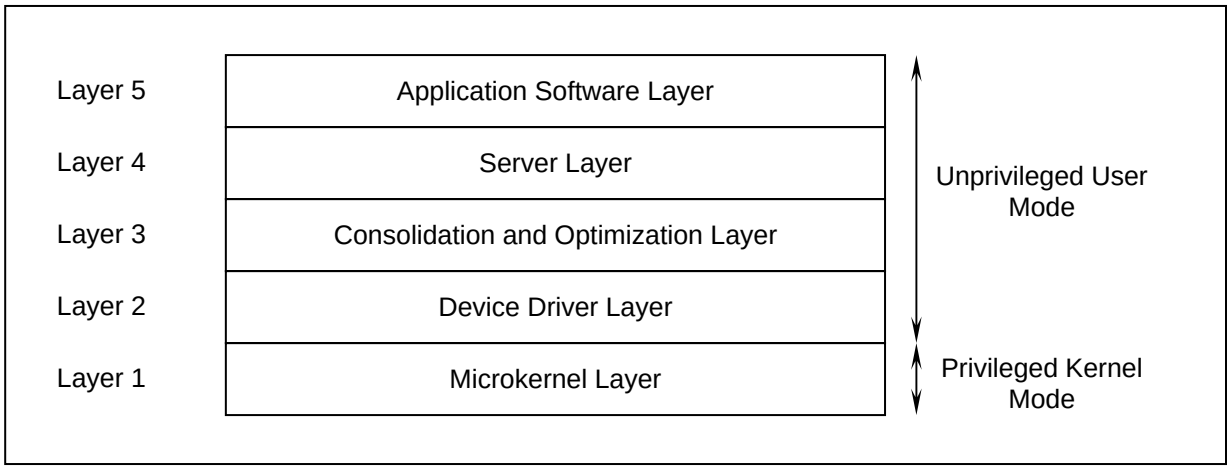

Fig. 1. The block diagram of the operating system showing the five different layers.

The entire system can be viewed as an assemblage of modules each of which may be either a system component or an application program. These modules use the message passing mechanism to communicate with each other and thus avail the services provided by one another. However, a module can initiate a communication with another module strictly according to the following three rules.

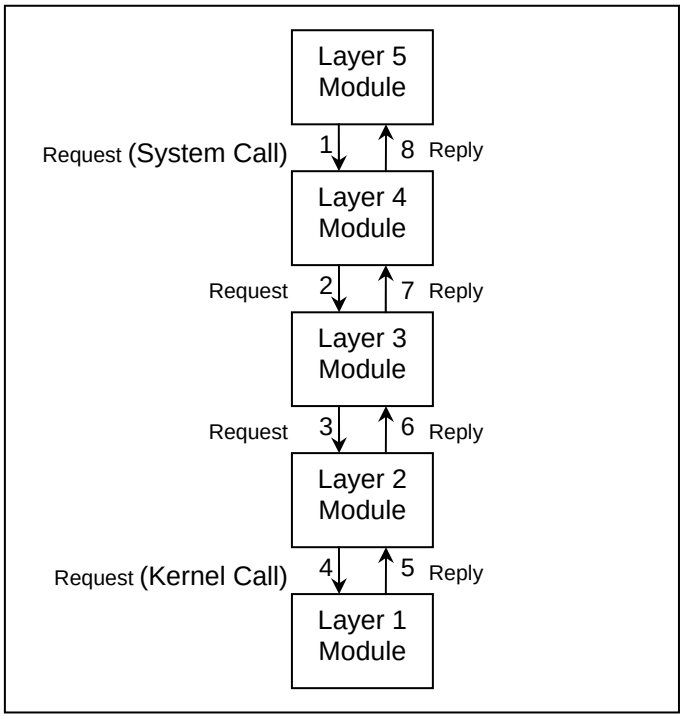

Fig. 2. A typical communication sequence among the modules illustrating a system call and a kernel call.

- **Rule 1.** A module M_1 can initiate a communication with another module M_2 in the same layer. They can pass any number of messages among themselves and the format of these messages may be mutually decided upon by them at the execution time.
- **Rule 2.** A module M_1 can initiate a communication with another module M_2 in the layer just below it by passing a 'request' message in one of the few predefined formats. On receiving this message, the module M_2 tries to service the request and sends a 'reply' message to the module M_1 again in one of the few predefined formats. If M_1 is a module in the Layer 5 then the 'request' message is known as a system call. Alternatively, if M_2 is a module in the Layer 1 then the 'request' message is called a kernel call by some researchers [7,9,10]. Fig. 2 illustrates the use of a system call and a kernel call in a communication sequence among the modules.
- **Rule 3.** A module M_1 cannot initiate a communication with another module M_2 that is not in its layer nor in the layer just below it.

Layer 1

The Microkernel Layer of the operating system consists of the microkernel and the clock driver. The microkernel addresses interrupt handling, interprocess communication and CPU scheduling. The microkernel also provides a comprehensive set of kernel calls using which the device drivers can indirectly access the memory and the input/output ports. The clock can be considered to be the most fundamental device in the system as its services are availed even by the microkernel. So, the clock driver is included in the Microkernel Layer only and it shares the address space of the microkernel. However, the clock driver is implemented as a separate process. The Microkernel Layer is the only layer in the system that is executed in the privileged kernel mode.

Layer 2

Above the Microkernel Layer is the Device Driver Layer. Each device, like disc, terminal and printers, has its own driver that runs as a separate process in this layer. Each such process has its own address space. In a sharp contrast to most other

operating systems, the device drivers run in the unprivileged user mode and cannot directly execute the privileged instructions. However, they are free to make kernel calls and thus indirectly avail the services of the privileged instructions.

Layer 3

Above the Device Driver Layer is the Consolidation and Optimization Layer. This layer contains components to integrate and strengthen the system services provided by the microkernel and the device drivers. Such a layer becomes particularly important if the services of more than one device are required for a single task. The message type reader is an important component in this layer and it keeps track of the types of messages being passed among the different modules.

Layer 4

Above the Consolidation and Optimization Layer is the Server Layer containing a number of important server processes. The file server accepts and services requests pertaining to the files. The process manager handles all process related activities. The network server implements a complete TCP/IP protocol suite and facilitates data communications with computers with similar or dissimilar hardware and software platforms. The information server keeps track of miscellaneous information and it is used for the purpose of debugging. The verbose server provides explanations of the internal working of the operating system to the user programs on being probed. The reincarnation server is a special server and also the parent process of all the system components in Layers 2 to 4. If any of these system components crashes or fails to respond to the periodic pings, then the reincarnation server kills it and then restarts it from a copy on the disc.

Layer 5

Finally, above the Server Layer is the Application Software Layer. This layer contains numerous user programs that are allowed to make any number of system calls to any of the servers in the Server Layer.

THE NOVEL FEATURES

Apart from using the venerable principles of operating systems, the design presented in the current paper uses two novel features. They are those of the reincarnation server and the verbose server. The following two subsections briefly overviews these two concepts.

The Reincarnation Server

The concept of the reincarnation server has been recently introduced by some researchers [5,9,10,11] but in a much limited sense. The design presented in the current paper tries to explore the scope of this novel concept. The reincarnation server facilitates automatic repairing of common failures of the operating system like transient failures and aging bugs. As already mentioned, the reincarnation server is the parent process of all the system components in Layers 2 to 4 and hence it can detect any crash of either of these components. Additionally, the reincarnation server can periodically send 'status request' messages to the components that are most susceptible to failures. If no reply is received from a component within a preset timeout interval then the reincarnation server replaces that component with a fresh copy from the disc.

The Verbose Server

As one of the requisites of being pedagogical, the operating system has an option of verbose operation. This option, when turned on, explains the internal working of the operating system to the user in a step-by-step manner. Such a verbose mode can be best realized using the verbose server. The verbose server maintains a log of the messages being passed among the different modules in the system and also collaborates with the information server to create a portrayal of the operation of the entire system. This portrayal, or a part of it, is available to any user program on demand. The verbose mode is realized by an application program, called the explanation module, constantly probing the verbose server and explaining the replies of the same to the user (Fig. 3).

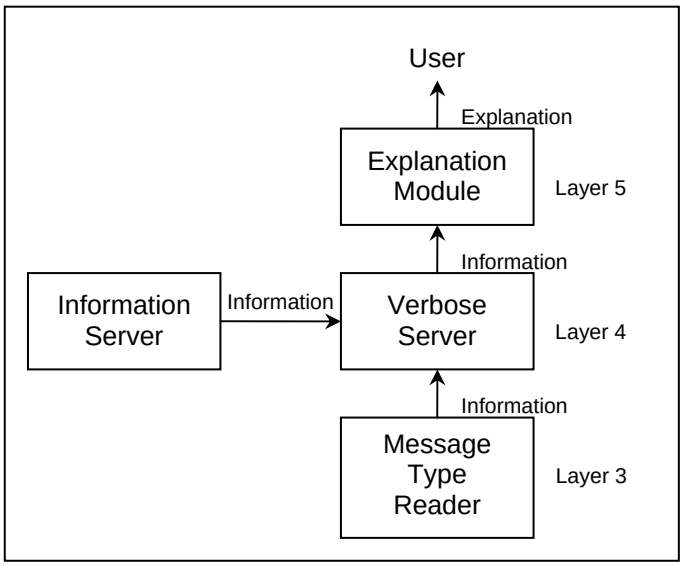

Fig. 3. Implementation of the verbose mode.

CONCLUSION

The current paper develops the primal design of a pedagogical operating system. Due to its pedagogic nature, the operating system will be immensely helpful in teaching and studying courses on operating systems in various universities and other educational institutions. The design of the operating system is quite unique as it amalgamates the object oriented methodologies with the concept of the microkernel. Moreover, the design extends the scope of reincarnation server and introduces the concept of verbose server.

Continued on Page No.

Continued from Page No.

FUTURE SCOPE

The authors are working on the detailed design and the implementation of the design discussed in this paper. Furthermore, the design can be modified by rearranging the layers or customizing the assemblage of the modules in the operating system. The performance of such a modified design can be quantitatively compared to that of the original one.

REFERENCES

[1] A. S. Tanenbaum, "A Unix clone with source code for operating systems courses", *ACM Operating Systems Review*, Vol. 21, No. 1, 1987, pp. 20-29.

[2] P. Chakraborty, "A language for easy and efficient modeling of Turing machines", *Progress in Natural Science*, Vol. 17, No. 7, 2007, pp. 867-871.

[3] P. B. Hansen, "The nucleus of a multiprogramming system", *Communications of the ACM*, Vol. 13, No. 4, 1970, pp. 238-250.

[4] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Principles*, Seventh Edition, John Wiley & Sons, 2006.

[5] A. S. Tanenbaum, J. N. Herder, and H. Bos, "Can we make operating systems reliable and secure?", *IEEE Computer*, Vol. 39, No. 5, 2006, pp. 44-51.

[6] P. Chakraborty, and R. G. Gupta, "A structural classification and related design issues of operating systems.", In *Proceedings of National Conference on Methods and Models in Computing*, 2007, pp. 265-273.

[7] R. Meurs, *Building Performance Measurement Tools for the Minix 3 Operating System*, M.Sc. Thesis, Vrije University, Amsterdam, 2006.

[8] A. K. Jones, "The object model: A conceptual tool for structuring software", *Lecture Notes in Computer Science*, Vol. 60, 1978, pp. 7-16.

[9] A. S. Tanenbaum, and A. S. Woodhull, *Operating Systems Design and Implementation*, Third Edition, Pearson Prentice-Hall, 2006.

[10] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Modular system programming in Minix 3", *USENIX ;login:*, Vol. 31, No. 2, 2006, pp. 19-28.

[11] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Minix 3: A highly reliable, self-repairing operating system", *Operating System Review*, Vol. 40, No. 3, 2006, pp. 80-89.