The Object Model of the JNUOS Operating System

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

ABSTRACT
An operating system is called an object oriented operating system if it is developed using an object model. In an object oriented operating system, all important entities are modeled as objects of suitably defined classes. The current paper presents the object model followed to design and implement the JNUOS object oriented operating system. Twenty-three different classes have been defined to model the various constructs in the operating system. The operating system has been developed as an assortment of cooperating components arranged in five layers. These components have been classified in three categories and the interactions between them have been classified in six categories. Precise rules have been defined to govern these interactions to obtain a formal object model.

KEYWORDS
Operating system, object model, object oriented operating system, delegation, microkernel.

INTRODUCTION
Over the decades, the object oriented approach has emerged as an effective means to develop software. It has been used successfully to develop a large number of application software in widely varying domains. The object oriented approach has been also employed to develop system software and in quite a few cases substantial success has been observed. It has been established by empirical research that object models can be proficiently used to design and implement the intricate structures of compilers [1, 2] as well as operating systems [3]. An operating system is said to be an object oriented operating system if and only if it is developed following an object model [3]. Using the object oriented approach, an operating system can be designed and implemented as a set of objects of suitably defined classes. Every important entity in the system, whether hardware realizable or not, is implemented as an object. The operating system is obtained as a collection of cooperating objects interacting according to a set of predetermined rules. The current paper describes the object model followed to develop a simple object oriented operating system. First, a brief overview of the operating system has been provided. Then the object model of the operating system has been discussed in details.

THE JNUOS OPERATING SYSTEM
In this paper, the object model followed to design and implement the JNUOS operating system is being presented. The JNUOS is a pedagogical object oriented operating system developed for the JNUVM virtual machine following the design presented by Chakraborty and Gupta [4]. A multiple server microkernel based architecture [5] has been followed to develop the operating system. The operating system has a modular and stratified design with five distinct layers comprising of system components and user modules. The lowest layer consists of the microkernel and the clock driver. The second layer contains the drivers for some basic devices. The third layer contains system components to consolidate and optimize the services provided by the components in the layers below it. The fourth layer contains several server components each providing a particular type of service. The fifth layer contains the shell and the user modules.

The JNUOS operating system has been implemented in the C++ programming language. The C++ programming language has been preferred because it is an object oriented language [6] well known for its system software development capabilities [7]. To realize the different constructs of the operating system, twenty-three different classes have been first defined. All important entities in the operating system have been then modeled as objects of these classes. Standard object oriented programming features have been used to implement the operating system. The concepts of inheritance and delegation have been used to define the related classes. Both static polymorphism, in the form of function overloading, and dynamic polymorphism, in the form of runtime function dispatch, have been used. Constructors and destructors have been defined for most classes. Some of these classes even have overloaded constructors. The size of the source code of the JNUOS operating system is approximately 6 KLOC. This measure, of course, excludes the numerous benchmark programs written to validate the operating system.

The JNUOS operating system has demonstrated the feasibility of amalgamating the concepts of microkernel and object oriented programming. The operating system has been tested by issuing the shell commands in different permutations and under different circumstances, and the correctness of their operations has been verified. The overall behavior of the operating system has been also found to be satisfactory [8, 9].

THE OBJECT MODEL
The Components and the Interactions
The JNUOS operating system, as already stated, has been designed and implemented as an assortment of components and modules arranged in five distinct layers (Fig. 1). There are three types of components and modules in the JNUOS operating system as defined next.
Fig. 1. The object model of the JNUOS operating system.
4. **Invocation.** This is an interaction in which a non-class module calls another non-class module in the same layer.

5. **System Initialization Access.** This is an exceptional interaction in which an instance of a service providing class is delegated to a non-class module in a higher layer. There is a very limited use of this type of delegation and that also for only the purpose of system initialization.

6. **Implicit Hardware Access.** All class components in Layer 2 have implicit hardware access through the microkernel. The hardware access has been thus implemented because the microkernel used here is only a pseudo-microkernel as already mentioned.

**The Layer 1**

The Layer 1 is the lowest layer of the JNUOS operating system. It comprises of the microkernel and the clock driver. The clock driver is realized by the class component `clock`. It is used by the microkernel, the shell and the service providing class `is_event`. The microkernel is constituted primarily by two service providing classes `pcb` and `process_set`. The service providing class `pcb` realizes process control blocks. The service providing class `process_set` realizes set of processes in the system with the support of the service providing class `pcb`. A global instance of the microkernel, as a whole, is used by the process manager. Moreover, all class components in the Layer 2 have implicit hardware access through the microkernel.

**The Layer 2**

The Layer 2 comprises of the drivers for some basic input/output devices. The drivers for keyboard, text display, graphic display, mouse, hardware printer and virtual printer have been realized using the component classes `keyboard`, `text_display`, `graphic_display`, `mouse`, `printer_real` and `printer_virtual`, respectively. The classes `text_display` and `graphic_display` inherit from a abstract class `display` that models a generic display device. Similarly, the classes `printer_real` and `printer_virtual` inherit from an abstract class `printer` that models a generic printer. The keyboard driver is used by the teletype. The text display driver is used by the teletype, the explanation module and the service providing class `user`. The graphic display driver is used by the startup module and the about module. The mouse driver is used by the explanation module while the two printer drivers are used by the shell.

**The Layer 3**

The Layer 3 contains the component class `teletype` that realizes the teletype driver. It is used by the shell, the file server, the reincarnation server and the help module.

**The Layer 4**

The Layer 4 comprises of a number of component classes and service providing classes. The service providing class `is_event` is used to realize events in the operating system for the use of the information server and the verbose server. The
service providing classes input_file, output_file and user are used to realize input files, output files and users, respectively. The classes input_file and output_file inherit from an abstract class file that models a generic file. The input file is used by the service providing class user, the file server and the login server. Moreover, the input file is used for system initialization access by the startup module and the shell. The output file is used by the service providing class user and the file server. Pointers to the objects of the service providing class user are used by the login server. The component classes process_manager, information_server, file_server, login_server, reincarnation_server and verbose_server realize the process manager, the information server, the file server, the login server, the reincarnation server and the verbose server, respectively. A global instance of the process manager is used by all component classes in the Layers 2 to 5. A global instance of the information server is used by the process manager, the information server and the verbose server. A global instance of the file server is used by the login server and the shell. The shell also has an instance of the login server and uses a global instance of the reincarnation server. A global instance of the verbose server is used by the explanation module. Moreover, the reincarnation server has pointers to objects of all component classes in Layers 2 to 4.

The Layer 5

The Layer 5 comprises of the startup module, the shell, the explanation module, the help module and the about module. When the system is booted, the startup module is invoked which in turn invokes the shell. The shell operates using the objects of various classes as already discussed. The shell also invokes the explanation module, the help module and the about module.

A Critique

When carefully examined, four critical remarks can be made about the object model of the JNUOS operating system. Firstly, inheritance has not been illustrated in Fig. 1. The reason behind this is that all the parent classes are abstract classes without any possibility of instantiation and Fig. 1 depicts the interactions between only the actual objects. Secondly, operator overloading, a useful feature of object oriented programming, has not been used in this study primarily because it was not needed. It will be interesting see the use of operator overloading in some other object oriented operating system. Thirdly, the object model presented here is specific to the JNUOS operating system. However, it can be helpful in evolving a general object model architecture for operating systems. Lastly, some features used in this object model, like the non-class modules and the global objects, move away from the object oriented ideology when considered in its strictest sense.

CONCLUSION

The object model presented in the current paper was used to develop the JNUOS operating system following the design perceived by Chakraborty and Gupta [4]. This verifies the hypothesis that object models can be competently used to design and implement large and complex system software like operating systems. Moreover, the success of the JNUOS operating system proved the viability of object oriented operating systems.

FUTURE SCOPE

This study is expected to be helpful in the formulating a hypothesis of compiler-operating system co-design. In this future study, a workable computer installation is to be systematically developed from the bare computer hardware by bootstrapping object oriented compilers and operating systems.

ACKNOWLEDGEMENTS

It was actually late Professor R. G. Gupta’s idea to develop the operating system using object oriented techniques. The authors would like to express their sincere gratitude for him. The authors also acknowledge Anurag Dixit and Akhil Mittal for their help, in various forms, during the course of this study.

REFERENCES