Abstracting Devices Using the Client/Server Model

João Paulo Souza de Oliveira[★], *Luciano Piccoli*[★],
Rafael Bohrer Ávila[★] & *Antônio Augusto Fröhlich*[✠]

[★]Universidade Federal do Rio Grande do Sul
Instituto de Informática
91.501-970 - Porto Alegre - RS - BRASIL
Tel.: +55 (51) 316-6159 Fax: +55 (48) 319-1576
e-mail: {jpaulo,luciano,bohrer}@inf.ufrgs.br

[✠]Universidade Federal de Santa Catarina
Departamento de Informática e de Estatística
88.049-970 - Florianópolis - SC - BRASIL
Tel.: +55 (48) 231-9498 Fax: +55 (48) 231-9770
e-mail: guto@inf.ufsc.br

Abstract

In this paper we aim to present a new operating system architecture based on device abstraction. This architecture abstracts the devices from the operating system microkernel in the same way microkernels hide the machine dependencies from the operating systems and operating systems abstract resources from the other programs. To achieve this, we use the client/server model, in a such way that a device driver found in conventional operating systems is substituted with a server process. This approach increases system flexibility, adaptability, reliability. A prototype has already been implemented at *Aboelha* microkernel to prove the model feasibility.

Each server in the proposed architecture is internally divided in two parts. The first controls the hardware, while the second receives client requests and, after communicating with the first, sends responses to the clients. We use a task and two threads for this implementation in such a way that each thread corresponds to a part of the server. The communication between the two parts is done through the data segment of the task, which is shared by the threads. Semaphores are used to synchronize the access to the data. The communication with the client that makes requests to the device server is done through mail boxes, ensuring compliance to existing software that uses this mechanism.

Keywords: operating system, client/server model, software abstraction, device driver.

1 Introduction

Analyzing the history of computer science and, particularly, operating system research (Weizer, 1981) one will realize that abstraction has been one of their major issues. We believe that it has happened because abstracting is the best way of increasing software feasibility, and we aim to present a new operating system approach that just focuses abstraction. Hence, we propose to abstract the devices from the operating system microkernel in the same way microkernels hide the machine dependencies from the operating systems and operating systems abstract resources from the other programs.

To achieve that we use the well known client/server model, in such a way a device driver at conventional operating systems is equivalent to a server at this new approach. In the next two sections we present the advantages of this architecture. After that, we show the *Aboelha* microkernel, which was used as the development environment. Then, we present the implementation of the proposed model and show how it adapts to the microkernel-based operating system modern features.

2 The Client/Server Model in Operating Systems

The client/server paradigm is a relatively recent programming model, which aims to divide tasks through server processes and client processes (Vaskevitch, 1995). Using this methodology in the construction of operation systems allows system services to be provided by servers, which can be implemented as processes that run at user level or at system level. This approach is extremely important to system maintenance and system upgrade, because a great complexity of the operating system kernel can be transferred to external processes.

Each service of the operating system can be provided by a different server. The processes that submit requests to these servers are called clients. Differently from conventional operating systems, client processes usually do not deal directly with the kernel. Figure 1 illustrates client/server communication cosidering this approach.

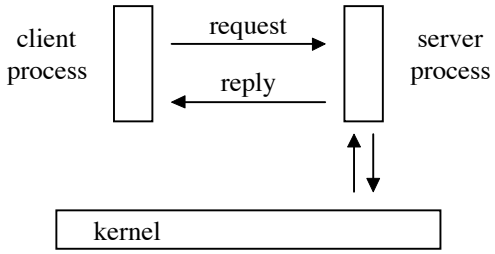

Figure 1 - Client/server communication

3 Abstracting Devices

Most of usual operating systems control the devices through device drivers, which in general are implemented inside the operating system kernel. In opposition, we believe that taking the device control out from the kernel we can improve system flexibility, adaptability and reliability, eliminating many of kernel dependencies on the machine. Hence, system implementation and maintenance can be done easier. This is also interesting to non-conventional environments such as embedded computing and user-like applications where operating system may need to interact to different kinds of devices. In this paper we show that using the client/server model to abstract devices from the kernel has many advantages. This model implies only in a new small server for each new device. These servers are called device servers.

We have used the client/server model to implement device abstraction because it effectively adapts to the device driver function. Drivers demand services to users in the same way servers do to their clients. Besides, client/server model is highly used and accepted by the community. Developers will request services from the devices as they do from any other server they interact with.

4 The *Aboelha* Environment

In order to prove the proposed architecture feasibility we have implemented a prototype in the *Aboelha* microkernel (Fröhlich et al, 1996a). It was used as the development environment due to its modern architecture and good features, which together provide all functionality necessary to the development of a distributed operating system (Tanenbaum, 1995).

Aboelha is a client/server object oriented microkernel that supports highly efficient process and memory management, communication and synchronization (Fröhlich et al, 1996b). To achieve that it uses five abstractions: tasks, threads, memory segments, mail boxes and semaphores. As shown in the next section, these facilities were extremely useful to the device server implementation.

5 Device Servers

A device server in this paper is equivalent to a device driver at standard operating systems, but its implementation is quite different. It makes use of many modern features as multithreading, shared memory, interprocess communication and synchronization.

Each device server is internally divided in two parts with different functions. The first part is responsible for dealing directly with the computer hardware, while the remaining part has the function of exchanging messages with client processes. Partitioning device servers avoids problems and confusion during implementation. It could also exploit the parallelism of multiprocessors. This division is shown in figure 2.

The division is done using multithreading, where a thread of the device server implements the first part of the server and another thread implements the second. These threads are called *executor* and *receptor*.

The executor thread deals directly with the peripheral and must have the correct knowledge about the use of the device. The receptor thread, on the other hand, has to manage all the requests sent to the device server and send them to the executor thread. Among its functions are request validation and authorization checking.

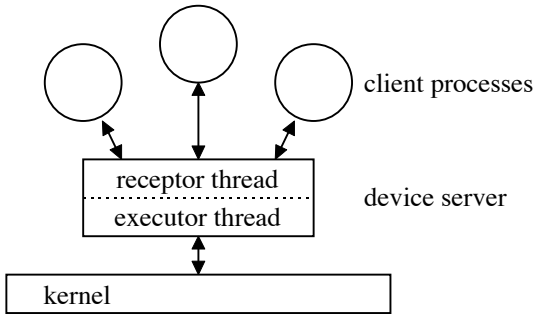

Figure 2 - Device server division

5.1 Device Servers Communication

Client processes communicate with device servers in the same way they do with all other servers, through mail boxes. This enforces system compliance with existing software.

In the other hand, information exchange between executor and receptor threads is done using a data segment, which belongs to the device server's task and is shared by its two threads. This approach ensures system performance. However, the communication must be coordinated, because it establishes a critical section (Silberchatz, 1990). In order to avoid inconsistencies created by critical sections, a semaphore is used. Figure 3 illustrates server internal communication.

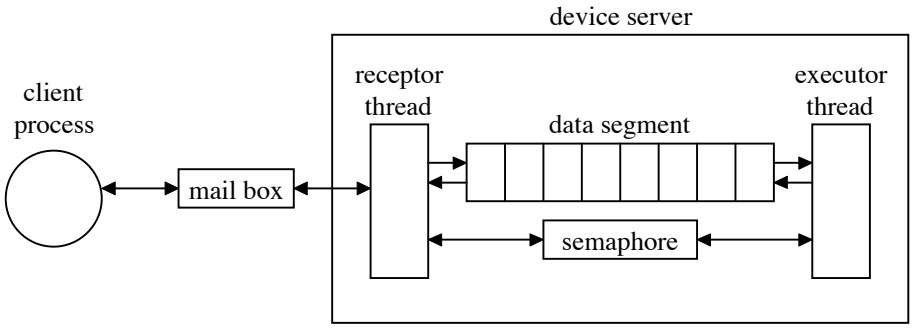

Figure 3 - Device server communication

5.2 An Implementation Example

One of the device servers implemented by the authors is the keyboard server. Its main goal is to enable system processes to obtain input data from the keyboard. According to the presented structure, the server is divided in two threads. The executor thread waits for keystrokes, analyzing them and storing the appropriate codes in a shared buffer. Concurrently, the receptor thread waits for requests from user processes. After receiving a new request, the receptor checks for data in the shared buffer and, if available, immediately send it as a reply to the client.

Figure 4 illustrates this situation. Client B sends a request to the keyboard server's mail box. The receptor thread receives the message, removes a code from the shared buffer and finally send it back to the client. At the same time, the executor thread waits and reads the keystrokes from the keyboard.

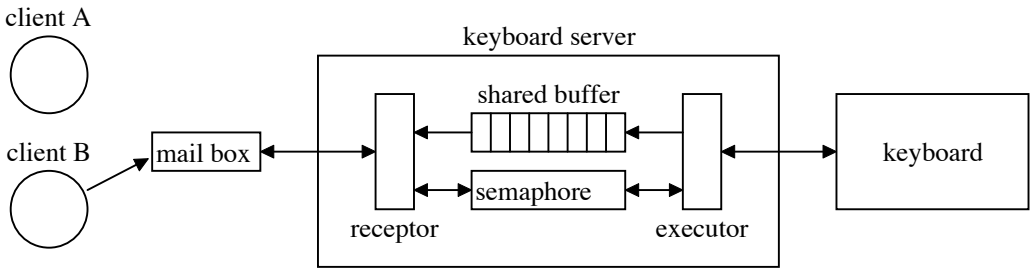

Figure 4 - Keyboard device server

6 Final Considerations

It has been a long time since research community has noticed operating system problems and concluded changes should be made in the way they were implemented (Lampson, 1971). But, although many software engineering efforts, we could not observe considerably evolution in this area.

We believe abstraction is a good way of changing this situation and present the client/server model as a way of abstracting devices from the operating system. After implementing this approach at *Aboelha* microkernel, it showed to be flexible, adaptable, reliable and feasible. Hence, this model can be used at next generation operating systems.

References

Fröhlich, Antônio A. M. et al. (1996a); Process Management in Nó//; In Proceedings of InterSymp'96, Baden-Baden.

Fröhlich, Antônio A. M. et al. (1996b); A Concurrent Programming Environment for the i486; In Proceedings of Information Systems Analysis and Synthesis, Orlando.

Lampson, B. W. (1971); On reliable and extendible operating systems; State of the Art Report, 1.

Vaskevitch, David. (1995); Estratégia Cliente Servidor; Berkeley, São Paulo.

Tanenebaum, Andrew S. (1995); Distributed Operating Systems; Prentice-Hall, New Jersey.

Silberchatz, Abraham et al. (1990); Operating System Concepts; 3 ed. Addison-Wesley, Massachusetts.

Weizer, N. (1981); A History of Operating Systems; Datamation.