Distributed Objects in Nó//

Jorge Ricardo Souza de Oliveira[1], João Paulo Souza de Oliveira[2] &
Antônio Augusto Medeiros Fröhlich[3]

[1] Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Engenharia Elétrica
Laboratório de Controle e Microinformática
88.040-900 - Florianópolis - SC - BRASIL
Tel.: +55 (48) 331-9422 Fax: +55 (48) 233-4729
E-mail: jorger@lcmi.ufsc.br

[2] Universidade Federal do Rio Grande do Sul
Instituto de Informática
91.501-970 - Porto Alegre - RS - BRASIL
Tel.: +55 (51) 316-6159 Fax: +55 (51) 319-1576
E-mail: jpaulo@inf.ufrgs.br

[3] Universidade Federal de Santa Catarina
Departamento de Informática e de Estatística
88.049-970 - Florianópolis - SC - BRASIL
Tel.: +55 (48) 331-9498 Fax: +55 (48) 231-9770
E-mail: guto@inf.ufsc.br

Abstract

This paper presents the object access control and distribution strategies for the Nó// (reads parallel node) project, which aims to develop a high performance, low cost, multicomputer and a concurrent programming environment for it (Corso, 1993).

Sharing resources is the main goal of multiprogramming environments (Andrews, 1990). Therefore, operating system must provide control mechanisms to make object access safety and efficient. Although there are well-known mechanisms for single memory systems, they are inadequate for distributed memory systems like Nó//.

Nó// operating system is projected to have an automatic and transparent computational load distribution. In this context, process and other operating system objects, like semaphores, mail boxes and still memory segments can freely migrate from a processor to another one. In order to allow that, without using inadequate broadcasting mechanisms (Tanembaum, 1992), Nó// operating system propose the indirect capability abstraction, which is presented in this paper.

The text first gives an overview of the whole project and shows how computing power can be enhanced by dynamic load-balancing system. Then, indirect capability concept is depicted to demonstrate how it allows security and transparent migration of processes and other operating system objects, improving system performance. In this context, it is discussed how Nó//'s process concept (Fröhlich et al, 1996) can be extended to allow distributed threads. Finally, it is presented a perspective analysis of distributed system evolution highlighting reliability, flexibility and efficiency.

Keywords: distributed objects, concurrent programming, operating system, parallel computing

1 Introduction

The Nó// project (Corso, 1993) aims to develop a comprehensive environment for parallel programming, including the conception of a multicomputer (Lücke et al, 1995). Project basic motivation is to adopt such a kind of architecture as a natural way to express parallel algorithms, which can be achieved as groups of processes that interact among themselves through message exchange. Project goals include, besides the multicomputer, the development of communication mechanisms (Fröhlich and Zeferino, 1996), an experimental operating system, a parallel programming language and several applications like parallel file system and compilers.

The project gathers several research groups in Brazil. At present, groups at Federal University of Santa Catarina and Federal University of Rio Grande do Sul are working cooperatively in areas like computer architecture, operating systems and programming languages.

To make use of all machine potentiality, it was developed an operating system specially for Nó//, the Aboelha operating system. Aboelha was conceived as a collection of servers supported by a micro-kernel, which runs in each of the multicomputer nodes. This micro-kernel, is responsible for the basis of memory management, processes, communication and synchronization. Other services, like file system, virtual memory and graphic interface can be achieved at user level by servers. In order to allow high performance, reliability, adaptability and flexibility the micro-kernel eliminates all machines dependencies with little overhead. Moreover, it is free from most operating system abstractions (Engler et al, 1994), like process hierarchy, the user concept and device drivers.

This paper focuses object access control and distribution strategies for Nó// project, describing how computing power and parallel programming expressiveness can be enhanced by transparent dynamic load-balancing system. After that, it is described some implementation details and personal conclusions are done.

2 Distributed Objects Architecture

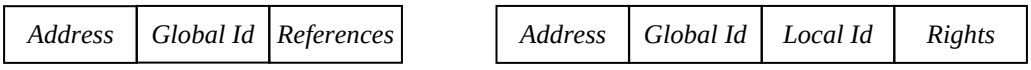
Sharing resources is the main goal of multiprogramming environments (Andrews, 1990). Therefore, operating system must provide control mechanisms to make object access safety and efficient. Although there are well-known mechanisms for single memory systems, they are inadequate for distributed memory systems like Nó//. At a multicomputer environment, objects may be shared by processes at different nodes. Besides, it is important to do a fair distribution of the objects among the multicomputer nodes, achieving an optimized computer power utilization.

The Aboelha operating system is projected to have an automatic and transparent computational load distribution. In this context, process and other operating system objects, like semaphores, mail boxes and still memory segments can freely migrate from a processor to another one. In order to allow that without using inadequate broadcasting mechanisms (Tanembaum, 1992), Aboelha proposes the indirect capability abstraction, which is presented in this paper.

Nó// multicomputer do not have any native broadcasting mechanism. That is achieved by multiple messages addressed to every multicomputer node. In this context, broadcasting can degenerate computing performance. The presented alternative for this is the indirect capability concept. This mechanism ensures that object localization is always known, even if the object, or the process that owns it, migrates. Then, broadcasting is not necessary.

Indirect capabilities are based on the traditional capabilities (Tanembaum, 1992) and beyond control object localization, they are used to identify objects and access rights. Indirect capabilities consists of two structures: *global capabilities* maintained by operating system and *local capabilities* owned by processes. A global capability references an object in the same machine or other global capability in another host. A local capability references a local or a remote global capability. These structures are shown in figure 1.

Considering a global capability, the *Address* field indicates the machine where the object is localized. The *Global Id* field identifies the object at operating system level. Finally, the *References* field indicates how many process are actually referencing the object through this capability. This last field is used to automatically release the object when it is not ever referenced by any process.



(a) Global Capability (b) Local Capability

Figure 1: Indirect capability's structures

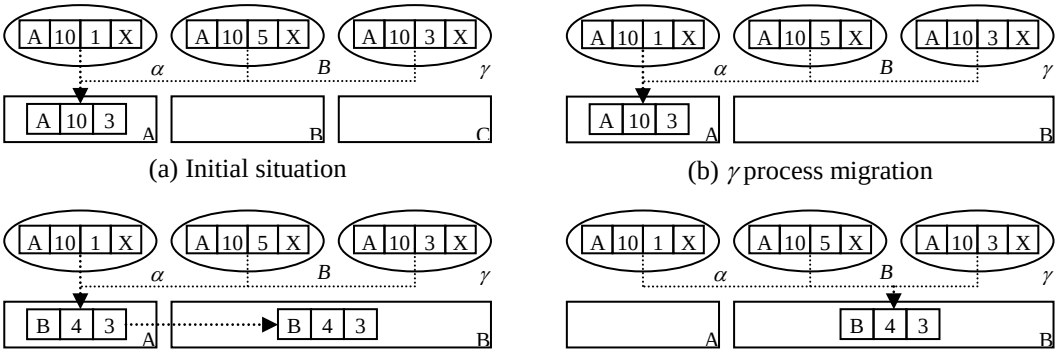
The *Address* and *Global Id* fields at local capabilities are similar to the correspondent fields at global capabilities. Although, these fields are not accessible by processes. Only the *Local Id* and *Rights* fields are accessible. *Local Id* identifies the object at process level and *Rights* controls the process access over the object. Next section shows how global capabilities and local capabilities are used to achieve transparent object migration

3 Object Migration Functionality

In order to understand object migration lets consider an example illustrating a common situation. Consider the α, β and γ processes running at the *A*, *B* and *C* machines sharing a system object. Figure 2a shows this situation. Realize that, despite all local capabilities refer to the same *Address* and *Global Id* values, the *Local Id* value may differ. Hence, the same object may be referenced at distinct ways by different processes.

Now consider that γ process migrates to B machine. This is shown in figure 2b. Realize that the local capability is still the same. Therefore, the process migration occurs transparently, without any re-starting mechanism.

Then lets consider the object migration to *B* machine. This event also occurs transparently, because the processes continue to reference the object through the global capability at *A* machine. This situation is illustrated at figure 2c. As the processes try to access the object, their local capabilities are updated and the *References* field at the global capability in A machine is decrement. When this field becomes zero the capability is released. Figure 2d illustrates this situation.



(a) Initial situation (b) γ process migration

(c) Object migration (d) Final situation

Figure 2: Object migration example

This reasoning can be applied to most kind of objects, like semaphores, mail boxes, files, etc. Besides, the indirect capability mechanism can be combined with a distributed shared memory strategy in order to allow distributed threads, as shown below.

An Aboelha operating system process is conceived as a combination of one task and one or more threads (Fröhlich et al, 1996). Tasks are passive entities and keep all process objects, like code segments and global data. Threads are active entities scheduled to run in a processor. They own a stack segment and a context of the processor registers.

Constant access makes memory a critical resource and, due to efficiency reasons, makes remote memory access unavailing. In this context, system performance can be degenerated whether threads are not placed at the same machine their tasks are. Notwithstanding, this paper proposes an efficient way to achieve thread migration, addressing the following issues.

Threads perform most memory access to code and stack segments. Hence, when a thread migrates to another host without its task, it must carry its stack segment and a copy of its code segment. This is possible, because the stack segment is private to each thread and the code segment is a read-only segment. Moreover, the thread can perform accesses to data segments through a distributed shared memory strategy, that works like a virtual memory mechanism.

4 Conclusion

The indirect capability approach can be used as an interesting alternative to the broadcasting strategy in systems that do not have native support to broadcasting. Besides, this approach may be used at a real time implementation of the Aboelha operating system. In this context, it is possible to have a real time system with transparent dynamic load-balancing.

The presented indirect capability mechanism also enhances programming languages expressiveness, since it is not necessary to express the object migration in the code. Besides, it is not still necessary to worry about details like how to re-start a process, or another operating system object, that has already migrated. Another contribution is the possibility of distribute threads of a same process over multicomputer nodes.

References

Andrews, G. (1990); Concurrent Programming; Benjamin Cummings.

Corso, T. (1993); Ambiente para Programação Paralela em Multicomputador; UFSC/CTC/INE.

Engler, D., Kaashoek, M. & O'Toole, J. (1994); The Operating System Kernel as Secure Programmable Machine; Proc. 6th SIGOPS European Workshop.

Fröhlich, A., Oliveira, João P. & Oliveira, Jorge R. (1996); Process Management in Nó//; InterSymp'96, 8th International Conference on Systems Research, Informatics and Cybernetics.

Fröhlich, A. & Zeferino, C. (1996); Process Communication in Nó//; Proc. International Conference on Information, Systems, Analysis and Synthesis, Orlando.

Lücke, H., Zeferino, C. & Silva, V. (1995); Um Multicomputador com Sistema Experimental de Comunicação; Proc. 7th Brazilian Symposium on Computer Architectures and High Performance Processing, Canela (pp. 137-150).

Tanembaum, A. (1992); Using Sparse Capabilities in a Distributed Operating System; Vrije Universiteit zu Amsterdam (technical report).