Building a Distributed Generic Layer for Multiple Personality Support on top of the Mach Microkernel

Franck Mével and Julien Simon[1]
Laboratoire MASI, Institut Blaise Pascal
Université Paris VI
4 place Jussieu
75252 PARIS CEDEX 05, FRANCE
Phone: +33 (1) 44 27 71 04
Fax: +33 (1) 44 27 62 86
{Franck.Mevel,Julien.Simon}@masi.ibp.fr

Abstract

Masix is a distributed multi-server operating system, based on the Mach microkernel, with multiple personality support. Its main feature is a *Distributed Generic Layer* (DGL), which offers distributed services to the personalities. The distributed multi-server architecture of Masix grants it a high modularity, but also raises many issues, which cannot be solved without adequate communication services. Thus, after a brief description of the structure of Masix, we study some of the features of the communication services offered by the DGL : location transparency, multi-protocol support, reliability, confidentiality and performance. Then, we describe a higher level feature of the DGL, i.e. migration support. Thanks to the definition of a new entity called *Generic Process*, our migration mechanism is transparent to the personalities, and induces minimal residual dependencies.

Keywords: distributed system, microkernel, Mach, personality, communication, migration

1 Introduction

In the past few years, the design of distributed operating systems has been moving towards a microkernel approach. Indeed, microkernels offer many advantages compared

[1]This work has been funded by the Institut Supérieur d'Electronique de Paris, 26 rue Notre-Dame des Champs, 75006 PARIS, FRANCE.

with monolithic systems, such as portability, modularity and dynamicity. In particular, the Mach microkernel [1] has been widely used to build many Unix-like systems, centralized like Lites [2] and Mach-US [3], or distributed such as Sprite [4]. Other operating system personalities have been implemented on top of Mach, like DOS [5] or OS/2 [6]. As far as we know, no distributed system has investigated the problems related to multiple personality support.

This paper describes the Masix[2] distributed operating system [7], based on the Mach microkernel, currently under development at the MASI Laboratory. Its primary goal is the simultaneous execution of multiple personalities, in order to run concurrently on a same workstation applications from the Unix, DOS, OS/2 and Win32 worlds. Masix also pools the resources of a workstation local area network (LAN), independently from the personalities that run on each node, thanks to a Distributed Generic Layer (DGL).

We first present the overall structure of the Masix system, then its most critical generic mechanism, i.e. communications, and finally its support for migration of user processes.

2 Overall structure

Masix is built according to the multi-server model, which grants it a high modularity. Indeed, each server can be written and tested independently from the others. Also, it is possible to add or remove dynamically a server to the system.

Masix is made up of two layers, shown on Figure 1. The upper one is the personalities, which implement the semantics of traditional operating systems. The lower one is the Distributed Generic Layer (DGL), which offers generic services to the multiple personalities (communication, shared memory, process management, fault tolerance, ...).

Masix is based on the Mach 3.0 microkernel. As such, Mach does not provide all the traditional functionalities usually found in an operating system. It only provides :

- task and thread management, which is the implementation basis for processes. A task is a resource allocation unit, whereas a thread is an execution unit ;
- local interprocess communication (IPC), according to a client-server model based on ports and messages. A port is an unidirectional communication channel, protected by send and receive rights. A message is a set of typed data, sent to a port ;
- physical and virtual memory management ;
- physical peripherals management.

3 The Generic Network Server

Many problems arise because of the modularity of Masix, such as the difficulty of keeping a global state of the system, because of its distribution among the microkernel and the

[2]The Masix homepage is located at **http://www-masi.ibp.fr/masix**.

Figure 1: Overall structure of the Masix System

various servers. As a consequence, many communications are initiated between servers running on a same node on the one hand, and between servers running on different nodes on the other hand. Thus, Masix needs adequate communication mechanisms to minimize their extra cost. They are provided by the Generic Network Server (GNS), and fit the needs of all the components of the Masix system, i.e. the DGL servers and the personality servers.

3.1 Location transparency

To simplify the design and implementation of Masix, it is desireable that remote IPCs follow the same semantics as the local IPCs provided by Mach. Thus, inter-task communications are based on the **mach_msg()** primitive, no matter where the tasks are located. This way, two communicating tasks do not have to make assumptions about their respective locations. This is all the more important since some mechanisms of Masix can move a running task from one node to another. One of them is migration, which we present in the next section.

To achieve this level of transparency, we extend the communication model of Mach by interposing the GNS between the tasks and the microkernel. A typical communication now takes place as shown on Figure 2.

1. task A looks up the name of task B. This name actually is a port ;
2. since B is not local (and has not communicated so far), its name is unknown to the name server. Assuming B is remote, the name server requests from the GNS

Figure 2: New communication model

a multicast to the name servers running on the network ;

3. the GNS multicasts the name resolution request ;
4. the name server running on B's node answers the multicast by sending the pair *(node number,port name)*, which is a unique global identifier for task B ;
5. the GNS stores this information, and forwards it to the name server. It also creates a proxy port for B, giving the receive right to itself and the send right to A ;
6. the name server delivers B's proxy port to A ;
7. A send its message on B's proxy port. The message is received by the GNS, which translates the proxy port into B's global identifier *(node number, port name)* ;
8. using its own protocol, the GNS sends the message to the GNS running on B's node ;
9. the GNS forwards the message to B.

This mechanism may seem very costly, but it happens only once, since B's global identifier is now cached by the GNS running on A's node. Furthermore, the following steps can be optimized :

- step 4 : instead of replying only to the GNS running on A's node, the name server running on B's node could multicast B's global identifier to every node. This way, B could be reached by any task without further name resolution ;
- step 8 : in the same way, the GNS running on A's node could multicast A's global identifier to every GNS, so that A could be reached quickly by every task, and particularly by B, which is very likely to reply to A. To speed up B's answer a little more, the GNS could also create a proxy port for A as soon as its global

identifier is known. This way, when A's message would reach B, everything would be set up for a quick answer.

3.2 Multi-protocol support

Since Masix is a multi-personality system, the GNS manages the protocols specific to each personality (TCP/IP, IPX, . . .), because it has to be aware of the state of every communication initiated by the applications. Indeed, the termination of an environment demands that all connections based on its network protocols be closed. It is thus mandatory that every protocol used by a personality is known to the GNS. For this purpose, the server enforces on the protocols the use of a dynamic registration/unregistration interface.

3.3 Reliability

A distributed system running on a LAN is vulnerable to node faults and to communication links faults. This issue is all the more acute in Masix that the system is made up of multiple servers, each of which might fault. This is the reason why a server can be replicated, to guarantee its availability. Thus, Masix includes reliable group communication primitives [8, 9], which are in charge of fault detection and recovery, as well as maintaining the coherency of replicated data. We are also investigating the possible use of failure detectors [10].

3.4 Confidentiality

In Masix, traditional network security issues take a new dimension. Indeed, servers exchange private information through the network. Were this information to be eavesdropped or altered, the security and the integrity of the system could be severely compromised. This is the reason why the GNS provide cryptographic services to the personality servers. Furthermore, thanks to its mandatory registration interface, the GNS controls the access to the physical network. Since unregistered protocols are denied access to the network, a misbehaving application is prevented from using its own protocol.

3.5 Performance

Most measures indicate that microkernel-based systems have worse networking performance than monolithic systems [11]. One might then conclude that microkernels are inherently unfit for distributed architectures. On the contrary, [12] show that a microkernel-based system can match the networking performance of a monolithic system. This is achieved by splitting a protocol functionalities between the network server and a library mapped into the applications' address space, as shown on Figure 3.

The library contains the performance-critical services, such as sending and receiving data. This prevents numerous and costly context switches between the applications and the server. The latter only contains the services which require its direct inter-

Figure 3: Protocol architecture

vention, such as opening and closing a connection. We are currently working on this decomposition.

4 Migration mechanism

4.1 Related work

Few studies have been led on migration on top of the Mach microkernel. [13] has worked on migration of Mach tasks. Task migration is implemented on top of Mach, but outside of any operating system emulation server. The major advantage is that the migration mechanism is totally transparent to the applications running on any operating system emulated on top of Mach.
However, since the Mach task migrates while the personality process remains on the source node, all system calls must be redirected to that node. This induces strong residual dependencies, which is a major drawback.
Nevertheless, this solution is based on an extended version of Mach, which supports network IPC and distributed shared memory [14]. These extensions make system calls forwarding easier. Furthermore, this approach is interesting in the case of tightly coupled multi-processor machines which use fast and reliable communication links, but may not be relevant in the case of a LAN where communications are costly and subject to failures.

4.2 Goals and definitions

We propose a migration support mechanism independent from the operating systems personalities, and we define a new execution entity, called Generic Process (GP). Each GP is tagged by a unique global identifier and corresponds to a Mach task, possibly multi-threaded. The only restriction is that a GP can migrate only to the nodes which run the personality of the corresponding process. Thus, we define two classes of user processes, the sedentary processes and the nomad processes :

- Sedentary processes will never migrate, because they are tied to the node's hardware (like the X server or the network daemons) or because they require frequent interactions with the user (like the shells). They are spawned by their personality process server and are only known by the node they were created on.
- Nomad processes may migrate to another node. They are spawned by the Generic Process Server (GPS) on the behalf of a personality and are associated to a GP. Since the DGL handles the processes' movements, migration becomes transparent to the personalities. This means that system calls need not be forwarded to the processes's birth node, except those dealing with user interactions.

4.3 Creation of a nomad process

The creation of a nomad process takes place as shown on Figure 4 :

Figure 4: Generic process creation

1. on node 1, personality A requests the creation of a new nomad process. As a consequence, the GPS creates a GP ;
2. this creation is propagated to every node of the network, so that each GPS may be aware of the new GP. This way, it could later migrate to any node, even if they did not run personality A prior to its creation ;

3. the creation of the new nomad process is multicasted to every occurrence of personality A running on the network. They create local copies of the process' ports, which will be used in case the process moves to their node.

Since the ports used by the personality servers to communicate with the process are available locally, and since the servers periodically receive updates on its state, every node running that personality is in a position to handle it. Thus, the migration of a nomad process is transparent to its personality.

This mechanism limits residual dependencies, since most of the system calls can be processed by the local personality servers. Only a few have to be redirected to the home node.

4.4 Migration as a Load Distribution and Fault Tolerance Mechanism

The migration decisions are taken by the GPS, according to a multi-criteria algorithm, based on CPU time, IPCs, and so on. The major issue is to collect the corresponding information, which are disseminated in the microkernel and the servers. Each generic server manages a fraction of this information :

- the GPS knows the CPU time consumed by a GP ;
- the GNS counts the messages sent and received by a GP ;
- the Generic File Server knows the number of open files and their respective physical location.

Periodically, the GPS polls the other servers to collect the relevant information. It has thus a global picture of the processes' state and may then take migration decisions. We plan to implement and test several migration algorithms. We are also investigating the possibility to use a different migration policy for each personality.

The Masix system can also initiate the mandatory migration of a GP. This can be very useful when a node has to be stopped, for instance because of maintenance operations. This way, long-lived applications may be moved to another node without losing the work accomplished so far.

5 Conclusion

We have presented the structure of the Masix distributed operating system, and particularly some features of its Distributed Generic Layer, which offers distributed services to traditional operating systems personalities. We have detailed some of the requirements of the Generic Network Server, which is a crucial part of the system. We have also proposed a transparent migration mechanism that may be used as a test platform for migration policies.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A New Kernel Foundation For UNIX Development. In *Proceedings of the USENIX 1986 Summer Conference*, June 1986.

[2] J. Helander. Unix under Mach - the Lites Server. Master's thesis, Helsinki University of Technology, December 1994.

[3] J. Stevenson and D. Julin. Mach-Us: Unix on Generic OS Object Servers. In *Proceedings of the 1995 Usenix Conference*, January 1995.

[4] Michael D. Kupfer. Sprite on Mach. In *Proceedings of the Third Usenix Mach Symposium*, April 1993.

[5] G. Malan, R. Rashid, D. Golub, and R. Baron. DOS as a Mach 3.0 Application. In *Proceedings of the 1st Usenix Mach Symposium*, pages 27–40. Carnegie Mellon University, November 1991.

[6] J. Phelan, J. Arendt, and G. Orsby. An OS/2 Personality on Mach. In *Proceedings of the 2nd Usenix Mach Symposium*, pages 191–201, April 1993.

[7] R. Card, E. Commelin, S. Dayras, and F. Mével. The MASIX Multi-Server Operating System. In *OSF Workshop on Microkernel Technology for Distributed Systems*, June 1993.

[8] A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 561–568, May 1993.

[9] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, 1994.

[10] T. Chandra and S. Toueg. Unreliable failure detectors for asynchronous systems. In *Proceeding of thr 10th ACM Symposium on Principles of Distributed Computing*, pages 325–340, August 1991.

[11] C. Maeda and B. Bershad. Networking Performance for Microkernels. In *Proceedings of the 3rd Workshop on Workstation Operating Systems*, April 1992.

[12] C. Maeda and B. Bershad. Protocol Service Decomposition for High Performance Networking. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 244–255, December 1993.

[13] Dejan S. Milojicić, Peter Giese, and Wolfgang Zint. Experiences with Load distribution on top of the Mach Microkernel. In *Proceedings of the 4th USENIX SEDMS Symposium*, San Diego, September 1993.

[14] B. Bryant, A. Langerman, S. Sears, and D. Black. Norma Ipc: A Task-to-Task Communication System for Multicomputer Systems. Technical report, Open Software Foundation, October 1993.