ABOS - an Agent-Based Operating System[1]

Mikael Svahnberg, Paul Davidsson, and Håkan Grahn
Department of Software Engineering and Computer Science
University of Karlskrona/Ronneby, Soft-Center, S-372 25 Ronneby, SWEDEN
{msv, pdv, hgr}@ipd.hk-r.se

1. INTRODUCTION

Modern operating systems should be *extensible* and *flexible*. This means that the operating system should be able to accept new behaviour and change existing behaviour without too much trouble and that it should ideally also be able to do this without any, or very little, downtime. Furthermore, during the past years the importance of the network has increased drastically, creating a demand for operating systems to function in a distributed environment.

Since agents are autonomous and can potentially adapt over time we argue that flexible operating systems can be realized by using agents in the kernel. The ability to easily communicate and coordinate work according to well-specified standards is something that we believe operating systems can benefit immensely from.

Further, the trends in the operating system community today seems to be focused on various schemes for supporting objects and object infrastructure for migration, persistence, and such. The overall aim of this research is to achieve flexibility, maintainability, and extensibility. However, the focus seems to be more in *how* to support the infrastructure rather than *what* to do with it once it is in place. This paper attempts to bridge this gap by presenting a design solution of an operating system kernel, called ABOS (Agent-Based Operating System), utilizing the ideas in such an infrastructure.

2. GENERAL LAYOUT OF ABOS

ABOS consists of a set of small and autonomous modules, or agents. Unlike traditional kernels where all modules share the same memory space, the agents in ABOS are run as separate processes. This yields great opportunities to modify and extend the behaviour of the system without having to recompile or even restart the operating system. ABOS is structured as a set of layers around the core (see Figure 1). The level of the services provided increases for each layer. The hardware is accessed wherever it is feasible, not forcing calls to propagate through more layers than necessary.

The *core* provides basic memory, process, and communication management. There is no intelligent behavior in this part, and its sole function is to act as an interface to the most central hardware, such as the

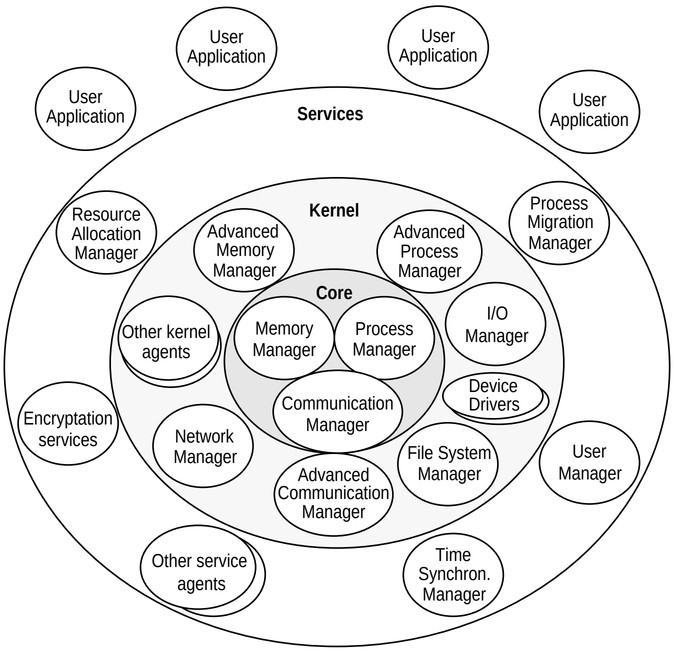

Figure 1. The layout of ABOS

1. For further details, see: Svahnberg, M. "*Background Analysis and Design of ABOS, an Agent-Based Operating System*", Master Thesis, University of Karlskrona/Ronneby, August 1998, http://www.ipd.hk-r.se/~msv/thesis/ABOS.pdf

CPU and the MMU. Outside of the core, the *kernel* provides the algorithmic behaviour for the core. Each of the managers in the core can have several corresponding managers at the kernel level, to provide application-tailored intelligence for their particular task. Device drivers and file systems are also managed at the kernel level. The *service* layer provides additional functionality that are not part of the kernel but still part of the operating systems, such as process migration and user management. Finally, *user applications* are run on top of the service layer.

3. EVALUATION

Looking at the performance of ABOS, we see that the number of context switches increases, as a result of the increased number of IPC calls. However, in contemporary machines the context switch time is so small that we believe that it has very little impact on the performance.

A comparison of the design of ABOS with that of a microkernel yields some similarities but also a number of interesting differences. A traditional microkernel operating system provides minimal functionality, much like the core layer of ABOS, and lets the user applications implement the rest. In ABOS this extra functionality resides in the kernel, which allows other applications to share the same kernel modules. This, in turn, allows for more intelligent behavior in the kernel modules for example when scheduling requests. Furthermore, the modules are run as autonomous units which enables mobility and easy plug-and-replace of functionality.

The main goal of ABOS was to increase flexibility, and this, in our humble opinion, has been achieved to a greater extent than we could have hoped for. System functionality, being implemented by autonomous agents in the kernel, can be replaced and extended transparently for the user processes that are currently running in the system. This gives an extremely high degree of flexibility while at the same time retaining an equally high degree of availability. Moreover, and this we find very interesting, you can install a new release of the operating system without even having to restart the computer!

A particularly interesting part of ABOS is the file system. Each file is embedded in an agent, through which all calls to the file are filtered. This file agent can also, as all agents, react in respect to its environment, and thus update the file or notify other entities on the system when something happens. By embedding each file in an agent, the files are given more responsibility and a possibility to influence the way they are stored and handled. Among the benefits of this solution are the possibility to migrate and replicate, which improves performance. Security, including access control and encryption, can be tailored for every file. The files can even present different views to different users. The most obvious disadvantages are that the files increase in size to be able to carry around its code and state, and that a file access will result in more code executed in order to handle the specific behaviour of a file. A prototype of the file system has been implemented, and the flexibility and benefits discussed above are confirmed by this implementation.

4. CONCLUSIONS

Traditionally the structure of the operating system is static, and applications need to adjust to this structure. In some cases this implies a trade-off between what you want to do and what the operating system allows. ABOS allows developers to adjust the operating system kernel to the applications running on it, while still being able to run more than one program concurrently. Also, ABOS has the potential to deliver new and more powerful services than traditional operating systems, such as those provided by the file system.

We are convinced that future operating system kernels will be more modularized and customizable than they are today. Our work shows that agents provide a promising way to achieve this.