Extendible Persistent System

ERDAL KEMIKLI and NADIA ERDOGAN
Department of Control and Computer Engineering
Istanbul Teknik University
80626 Ayazaga, Istanbul
TURKEY
erdalk@kocsistem.com.tr, erdogan@cs.itu.edu.tr

Abstract: The backlog in software development, percentage of unsuccessful projects and high costs of new software forced the industry to use the term "software crisis". Increasing complexity of the computer systems necessitates new system paradigms, so that human cognitive capacity will be able to comprehend and maintain.

Persistent Operating systems will be the next logical step in the higher level abstraction of electronic information systems. They are an attempt to reduce the complexity of programming and system design. This paper summarizes the research on the Extendible Persistent System (EPS) and explains briefly requirement and design of this system.

Key-Words: Persistent system, operating system.

1 Introduction

A program creates and manipulates a large amount of data throughout its execution. Each item of data will have a different lifetime [1]. While programming languages provide excellent, integrated support for transient data, other categories of data is supported by Database Management Systems, or file systems.

In a conventional operating system two abstractions of data access and storage are provided: virtual memory and files [5]. While we can directly access data in virtual memory, data in files can be accessed using system calls. A persistent system, which differs from conventional systems, has a different abstraction of data storage environment unifying virtual memory and the file system to provide a single abstraction o data.

The idea behind persistence is simple [2]: all data in a system should be able to persist (survive) for as long as that data is required. In this sense persistent systems provide a uniform abstraction over storage, whereas in contemporary systems long lived data is treated in a fundamentally different manner from transient data.

PS-ALGOL [3] coined the term “orthogonal persistence”, that is, the possibility that any object can be made to persist, independent of its type or the way it is used in the program. Systems,which provide orthogonal persistence, treat all data identically as persistent objects.

There are certain advantages of the persistent systems over the transient ones [4]:

- Several programs or different runs of the same program that use data with the same structure do not have to build the structure from scratch every time.
- The programmer is saved from task of writing extra code for I/O operations to transfer the data to a file and then transfer it back into the memory.
- Since data objects can persist, a program that uses the data doesn’t have to run to completion once it starts; the execution can be temporarily distributed.

This paper briefly describes the requirements and design of such a system, Extendible Persistent System (EPS). The goal of this system is to demonstrate an extendible and tailorable computing system model, which supports persistent object paradigm. The EPS will be suitable to be used as a base for an expandable system with the required server functionality. EPS supplies facilities that will ease developers to extend the functionality of the system with simpler programming. Moreover, the resulting extended system will not need extra system administration tasks and complex configuration management.

2 EPS Requirements

EPS is aimed to be a system of easy programming and use, supposed to work on existing architectures and operating systems. In this section, requirements of such a system are explained.

2.1 Programming Interface Requirements

Programming under EPS will not be very different than a conventional system. The well-known programming languages are used with few but critical additions to them. This will enable programmers to develop new systems based on EPS with a short learning period. Consequently, this harmony with the existing systems will facilitate the widespread use of EPS.

The loading of persistent objects are handled by explicit function calls. While this looks like an extra task to programmer, we shall be aware of the fact that the programmer shall declare the binding between the persistent object and the database in any case.

The programmer decides the loading time. This gives flexibility to design programs for different purposes and characteristics.

Type checking will be conducted during the compilation phase. It will also be possible to delay the type checking to the execution phase, to enable programmers develop code for currently non-existent persistent objects. This will be made available through a compiler option.

A default persistent object called as the 'program table' will be available to every program, and the necessary code to handle this persistent object will be inserted automatically to the program during the link phase. This object will include standard fields such as 'variable name', `variable value'. There will be extra functions to retrieve and modify the values in the program table. This table is planned to be used as the practical storage area for every program. The structure itself can be used in the implementation of an RDBMS-like server.

2.2 Security Requirements

There are two basic mechanisms to access objects: capabilities and access lists[8]. Capabilities are non-replicable tickets those give authorization to their owner to access an object. Capabilities can be used to name, protect and define the operations on that object. General characteristics of capabilities:

- If a process owns a capability, then it has the access right to the related object.
- Capabilities allow sharing of the objects.
- Capability should be non-replicable and non-decipherable. A system that lets users to create capabilities is useless.

Very little work has been done on the removal of already distributed capabilities. Amoeba system [9,10] with the most advanced features can remove the capability by modifying the random number of the capability, but partially removing the capabilities is not possible.

The EPS system shall provide the following capability features:

- It shall support resource sharing. By passing a capability to a new object the access rights of the related object shall be transferred.
- While capabilities are transferred, it shall be possible to restrict the access rights . A client shall be able to transfer a restricted copy of a capability.
- Formerly distributed capabilities shall be removed from some of the clients or some of the rights in a capability shall be taken be back.
- It shall be possible to remove a formerly given capability completely.

2.3 Architectural Requirements

The mechanism to support these features shall have the following architectural constraint: There shall be no need for a protected storage area. A constraint on this feature will restrict the number capabilities.

Using current operating systems, the implementers of a persistent system must manage the address translation tasks. A persistent operating system will provide an abstraction consistent with our requirements as a fundamental building block. Intrinsically a persistent operating system would be capable of providing all the functionality of traditional operating systems. The research issues are those same issues which compromise the implementation of persistent systems when conventional operating systems are used as a platform, namely: addressing, resilience, process management and protection.

The major constraint is that the operating system should run on conventional architectures . The effects of this constraint are that on most current architectures addresses are a maximum of 32 bits long, there is no hardware support for object protection and the only memory management hardware available is based on fixed sized pages. Thus, in order to construct a persistent operating system on conventional hardware some compromises must be made.

3 System Design

EPS has a multi-tier architecture that is designed to extend naturally based on the needs. The basic system is composed of three basic components; Naming and Protection Server (NPS), Object Server (OBS) and Client object library (COB). An underlying messaging facility interconnects these three modules. While these modules are the base system, any user need can be satisfied by extending the system via active objects (AOB).

This modular system is based on an existing UNIX implementation, Linux. Linux, with its open source code and worldwide support by its user community was the most logical choice of operating system to use. The system language is chosen to be the C programming language because of its natural integration of UNIX operating system.

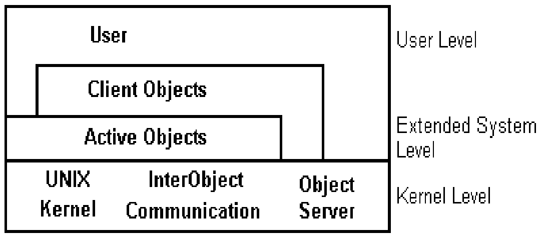

Fig. 1 - EPS Layers

3.1 Messaging System

To facilitate the usage, a high level interface matching with the general requirements of the system is provided. The primitives are designed to support object to object synchronous and asynchronous communication need. Unix IPC/RPC mechanisms is used as the basic mechanism. This high level primitive is used both in systems development and application development.

3.2 Object Server

OBS can be logically considered as a part of the operating system kernel. Physically it is implemented as a user level module. Object server loads active or passive static objects. While active objects consist of data and methods, passive object contains only data but do not have methods to modify it. Static object is an object, which resides in the long-term memory and needs an extra operation before being used.

3.2 Naming and Protection Server

NPS is implemented in the form of a server process. This choice makes it possible to include an indefinite size object table to perform some of the functions. NPS is responsible form the security and synchronization of objects.

InUse table (which itself is a persistent object) in NPS is used to keep the records of all currently used/loaded objects. It actually holds the entire capability information. Since the number of loaded objects can not be anticipated, its size is dynamically adjustable.

3.3 Client Object Library

Client object library which is used by every EPS user program and AOB has facilities to hide the underlying complexity of the system. The majority of the system primitives are implemented fully or partially in the COB. The services of COB also include the local (in-process) implementation of synchronization and address translation facilities, which are transparent to the application programmer.

4 Conclusion and Future Work

EPS is one of the efforts in the scientific community for the design and development of a computer system with a different philopsopy [6,7]. The main concerns of EPS has become the software crisis and development/management of increasingly complex systems. These problems are planned to be solved applying an innovative technology.

Currently, explained EPS design is under development and hoped to be completed in a short time. Our experience during the design and development of EPS confirmed our belief on the usefulness of such a system.

The second phase of the research will include experimental studies for the determination of the effectiveness of the system in reducing system complexity and improving programmer productivity. These experiments will test the added value of the persistent paradigm and our implementation.

References:

[1] Atkinson, M. P. et al, An Approach to Persistent Programming, *The Computer Journal*, Vol. 26, No. 4, 1983, pp.360-365.

[2] Dearle A., Rosenberg J., Jenskens F.A., Vaughan F., and Maciunas K.J, An Examination of Operating System Support for Persistent Object Systems, *Proceedings of the 25th Hawaii International Conference on System Sciences*, Vol. 1. Editors V. Milutinovic and B. D. Shiver, IEEE Computer Society Press, Hawaii, USA, 1992, pp. 779-789.

[3] Atkinson M.P., Chisholm J., and Cockshott W.P., PS-Algol: An Algol with a Persistent Heap, *ACM Sigplan Notices*, Vol. 17, No 7, 1982, pp. 24-31.

[4] Sajeev A. S. M., and A. J. Hurst, Programming Persistence in X, *IEEE Computer*, Vol. 25, No. 9 , 1992, pp. 57-66.

[5] Tanenbaum, A.S., *Modern Operating System*, Prentice Hall, 1992.

[6] Shapiro M., Object Support Operating Systems, *Workshop on Operating Systems and Object Orientation at ECCOP-OOPSLA*, Position paper, 1990.

[7] Varhol P.D., Trends in Operating System Design, *Dr. Dobb's journal*, May 1994, pp. 18-27.

[8] Morrison R., Brown A.L., Connor R.C.H., Cutts Q.I., Kirby G.N.C., Dearle A., Rosenberg J., and Stemple D., Protection in Persistent Object System, *Security and Persistence*, Rosenberg J., and Keedy J.L. (editors), Springer-Verlag, 1990, pp. 48-66.

[9]Mullender S.J., van Rossum G., Tanenbaum A.S., van Renesse R., and van Staveren H., Amoeba: A Distributed Operating System for the 1990s, *IEEE Computer*, Vol. 23, No. 5, 1990, pp. 45-53.

[10]van Renesse R., van Staveren H., and Tanenbaum A.S., The Performance of the Amoeba Distributed Operating System, *Software - Practice and Experience*, Vol. 19, No. 3, 1989, pp. 223-234.