RHODOS --- A Microkernel based Distributed Operating System: An Overview of the 1993 Version

Damien De Paoli
Deakin University
26 PUBLICATIONS **165** CITATIONS
SEE PROFILE

Andrzej Goscinski
Deakin University
245 PUBLICATIONS **1,727** CITATIONS
SEE PROFILE

Michael Hobbs
Deakin University
44 PUBLICATIONS **625** CITATIONS
SEE PROFILE

RHODOS — A Microkernel based Distributed Operating System: An Overview of the 1993 Version*

D. De Paoli, A. Goscinski, M. Hobbs, G. Wickham

{ddp, ang, mick, gjw}@deakin.edu.au

School of Computing and Mathematics
Deakin University, Geelong
Victoria 3217, Australia.

Abstract

The current direction of computer systems is increasingly moving away from centralised system and moving towards distributed sets of workstations and personal computers connected by a network, known as distributed systems. This has led to the development of specialised operating systems to take full advantage of the resource sharing potential that a distributed system provides. The specialised operating systems being developed for these situations are known as Distributed Operating Systems. Research into this area is still at the primary stage with little firm evidence indicating what components of Distributed Operating systems are more important than others.

RHODOS is a microkernel based distributed operating system that has been designed and detailed from the study of a number of existing Distributed Operating Systems. The primary object of RHODOS is to enable the study and testing of all components that combine to form a distributed operating system and not just a limited few. This paper introduces the architecture of RHODOS, highlighting the benefits envisaged by our design and the extra research areas we consider important that are not being investigated by other builders of distributed operating systems.

*This work was partly supported by Australian Research Council under Grants A48831034, A49232429 and the Deakin University Research Grant 0504054151.

INDEX

1 Introduction

There are some indications that distributed operating systems will achieve their maturity in a few years time. One of these indications is work carried out by Microsoft, whose research division is led by Professor Rick Rashid, the leader of the Mach project.

However, an analysis shows that nearly all of the currently-existing distributed operating systems are experimental: none of the available systems are of commercial quality (with the possible exception of Mach, and QNX). These existing systems were developed on the basis of quite disparate approaches: their developers made different assumptions, and focused their attentions on different subjects.

They do not address such issues as the influence of the operating system architecture, the locations of different services within an architecture, and several methods used to develop different services, on the overall system performance. The trade off between the performance gains from providing such services as global scheduling supported by process migration and the costs of providing these services has not been addressed, and even some researchers question the feasibility of process migration in a real distributed system. User autonomy, lost in a distributed system, has been neglected and security aspects are not treated widely and deeply. User friendliness and quality of work of users, which could be improved by introducing, for instance, attributed naming have been neglected.

Despite claims that one system is faster than other systems, the performance studies supporting these claims are generally not satisfactory. Two problems arise in attempting to evaluate their work. The first is that it is difficult to make comparisons — especially performance comparisons — between such systems with such disparate design goals. Moreover, the designers of these systems have given little explanation as to how they arrived at their design decisions, and have provided few performance measurement results. Second, users of distributed systems can improve their efficiency and quality of work if they can access resources from other computational environments which very often are not compatible with their computers.

The above issues and the strive to carry out research into the theory of distributed operating systems resulted in a project concerned with the development of a ResearcH Oriented Distributed Operating System (RHODOS), which was to be both a high performance distributed operating system, and a test bed to study research, design and development issues related to this class of operating systems [Gerrity et al. 91].

We investigate and compare alternative contending structures and methodologies for implementing the components a distributed and open operating systems.

Building a system which allows both qualitative analysis and quantitative comparisons, it must sit on top of a bare machine. This provides access to all resources without interference from any part of an underlying centralized operating system. We decided that RHODOS should be developed on the basis of a distributed microkernel and kernel, only barest minimum of functions are supported within the microkernel: where the majority of these functions are provided by specialized kernel server processes; easy expansion and debugging is guaranteed; and base for heterogeneity, scalability and high performance is provided. Furthermore, we decided that user services should be provided by system processes.

This paper reports on the results obtained in the last two years of our project, i.e., research into the design and development of the Deakin version of RHODOS. To allow the reader to understand the reasons behind some of our work, Section 2 introduces aims and objectives of our design and design objectives. Section 3 details the design of the RHODOS microkernel including the services and objectives that it provides. Section 4 and Section 5 report on the system processes that go to form the kernel and system servers, respectively. Section 6 concludes this report and indicates the areas of current and future research for the RHODOS system.

2 Overview of RHODOS

2.1 Aims and Objectives

Our design approach has been to utilize as far as possible the successful concepts and solutions from existing research distributed operating systems, while proposing new solutions to deal with the less successful aspects and with unsolved problems.

We have designed and implemented a test bed to study design issues and to compare different solutions new and used (or suggested) for the development of components of distributed operating systems. We are studying:

- new architectures of distributed operating systems, including microkernels, kernel servers and system servers;
- services provided by a microkernel and the capacity of a system call library and its influence on the kernel servers and their cooperation;
- the ways for improving performance by employing global scheduling with adaptively changing components: load balancing and static allocation;
- load balancing, process migration, and computation and communication load data collection facilities and their cooperation and placement in the distributed operating system architecture;

- moving processes (process migration), process and memory management and interprocess communication facilities and their cooperation;
- improving the effectiveness and quality of user's work, through naming: in particular attributed naming, and object sharing using the concept of trading;
- providing user autonomy in both homogeneous and heterogeneous environments by employing attributed naming and resource trading;
- transparency in a distributed system with movable objects using naming, interprocess communication and process migration facilities;
- high performance communication using interprocess communication facility and transport protocols;
- fast transport protocols for both distributed systems and fast networks;
- improving communication security by employing software-based communication security facilities;
- improving user security by utilising one-way, two-way and conference authentication;
- relationship between interprocess communication and communication security facilities;
- high performance and reliable distributed file facilities supporting basic file operations, transactions and replication;
- performance measurements of distributed operating systems.

The RHODOS test bed has been developed in such a way that we can carry out experiments on the influence of different methods and algorithms on the overall performance of distributed system. The effects can then be compared of which parts of these facilities have the greatest influence and impact on the overall performance.

2.2 Design decisions

The following criteria were selected to guide the design process for RHODOS:

- A very clean separation has been maintained between Policy and Mechanism, to enable an efficient study of the effect on system performance of a variety of policy decisions utilizing a few alternative mechanisms.
- The use of a microkernel architecture for RHODOS was chosen to provide a solid foundation on which a flexible operating system can be built. This allows a researcher to easily carry out

comprehensive performance study, to easily port a system to other architectures and to generate application oriented distributed operating system.

- RHODOS has been constructed so as to allow user processes to have access to a freely-available pool of services, such as a file service, name service, etc. These are to be provided by a number of *server* processes which take requests from user processes and return the desired result. The user processes are thus clients to the servers.
- RHODOS should provide a multitasking and multiuser environment. Several users will be able to utilise each machine in the system, and the number of active processes may exceed the number of CPUs. Initially the system is composed of single-CPU homogeneous machines, but RHODOS is designed with a heterogeneous system in mind, including multiprocessor machines.
- A global scheduling policy to ensure an overall balanced system load will be based on load balancing (using process migration) and initial placement (using static allocation) of processes.
- RHODOS provides the basis for the implementation of resource management and protection functions.
- The communication subsystem of RHODOS is based on an efficient and fast transport protocol (RHODOS Reliable Datagram Protocol) which uses the services of IP and Ethernet for communication in the RHODOS environment. Thus, a specifically distributed system transport protocol (RRDP and others) can then be compared with DoD protocols implemented on the same platform (e.g., TCP/IP).
- Objects in the distributed system have attributed names for protection and identification purposes. Moreover, there is a hierarchy of names: User names, System names and Locations (Physical Names).
- The majority of RHODOS objects are totally distributable. They can be accessed from any node in the system, and may well migrate from node to node to improve system use. Therefore the location of each object has been made transparent to the user.
- RHODOS supports research into effective process migration and load balancing policies, by enabling close cooperating between the Migration Manager, Process Manager, Memory Manager and the IPC Manager in order to provide efficient, fast, and transparent migration. Both processes and teams of processes can be migrated.

2.3 Logical Design and Components of RHODOS

The logical organization of RHODOS is shown in Figure 1. There are three levels of processes supported under RHODOS, User Processes, System Servers and Kernel Servers. Each process executes in user mode and is confined to an individual address space which is controlled and maintained by a RHODOS microkernel, the Nucleus. The set of kernel and system servers are known collectively as System Processes.

The lowest logical level of processes within RHODOS are those allocated to perform tasks for the user. User Processes have no special privileges and obtain services and resources by calls to the Nucleus and System Processes.

System Servers constitute the next layer of process hierarchy. Similar to User Processes these have access to resources via the standard system calls. System Servers however, utilise privileged communication with Kernel Servers. For example, the Global Scheduler is the only process that can issue requests to the Migration Manager.

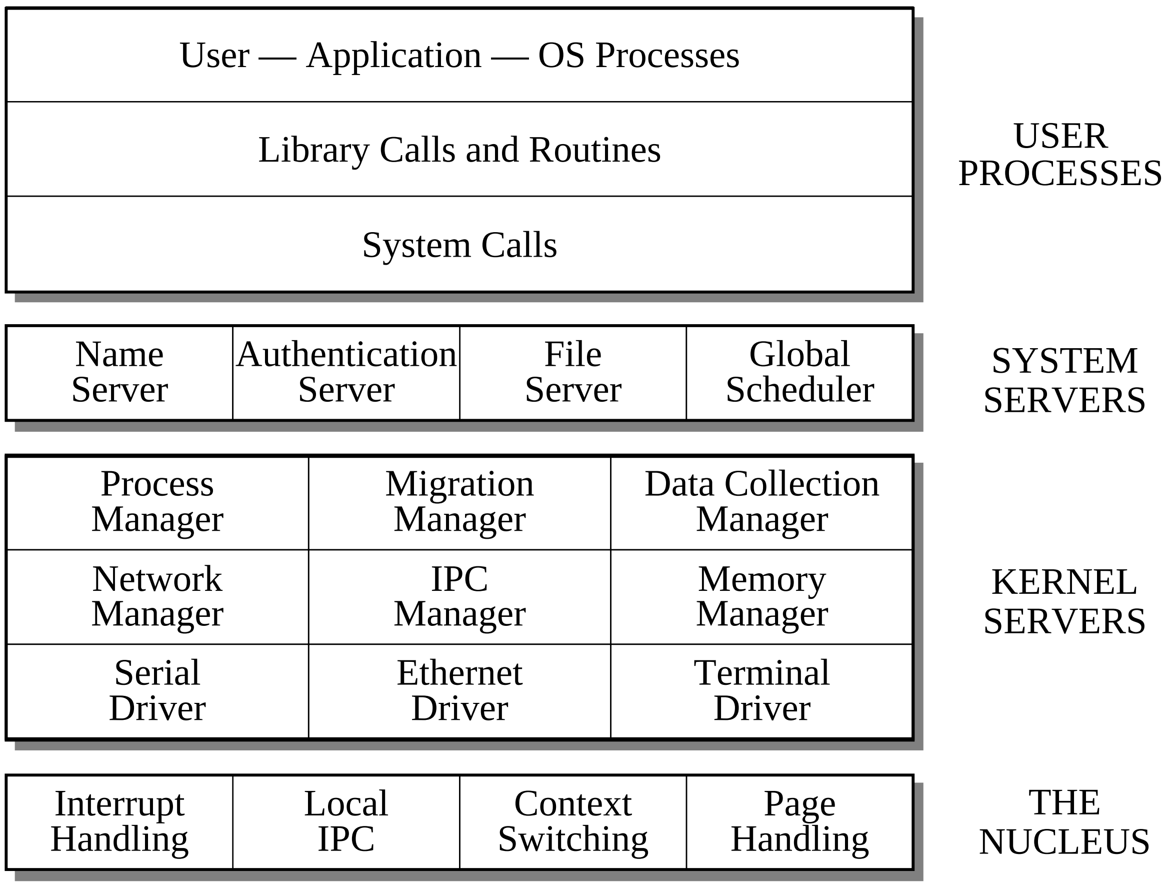

Figure 1: The Process Layers of RHODOS

The final layer provided within the RHODOS process hierarchy has the highest level of privileges. Kernel processes have the closest interaction with the Nucleus out of the system processes. In addition to standard system calls the Kernel Servers have access to *privileged* system calls which allow the modification of data buffers and structures within the Nucleus. To maintain consistency, locking and mutual exclusion techniques are used.

A point worth noting is the benefit obtained from system transparency. A System Server does not necessarily have to be located on the same host as the User Process, if the server is not found on the current host it is the responsibility of the system to forward the request onto the correct host. This enables a number of hosts to share the one System Server, whilst maintaining the impression to the User Process that the server is located on the local host.

2.4 Operational Concepts

The basic fundamental concept of the RHODOS system is a process. System processes of RHODOS cooperate based on the client-server model using messages. The exchange of these messages is accomplished by either message passing or the RPC paradigm.

Two classes of processes are utilised in RHODOS. The first class of processes are called heavyweight processes (Unix supports this class of processes) and which share code, however, maintain separate data and stacks. The second class of processes are called medium weight processes, which share both code and data - however, their stacks remain separate. The latter class was introduced to improve performance by use of sharing data.

The second fundamental concept of RHODOS is a space. A space is a continuous segment of memory which has a known start address and a known length. A process in RHODOS consists of three spaces: a text space, data space and a stack space. Although each block of memory has different attributes, they are all manipulated as spaces.

Each resource and service (object) in RHODOS has a system name (SName) used by the operating system and a user name (UName) used by users. A name server is responsible for resolution if a server maintaining an object is remote; and the mapping of user names onto system names. An object’s system name is a structure which contains: a user identification number, object type, indication of which copy of the object was created, the object’s local name, and the access rights and checksum (if the object is protected using capabilities).

In order to provide user friendliness, user names are attributed names, where each attribute provides a feature of an object. The decision to use attributed names made in 1987 has been right - the whole management of objects in open systems and in particular using traders is going to be based on attribute names.

3 Microkernel Architecture of RHODOS

Operating systems can be designed and built with a variety of differing architectural styles and forms. The primary styles used as a basis for operating system development include: monolithic, kernel and microkernel based. As introduced in Section 2.2, a microkernel based architecture was chosen for the development of RHODOS.

A number of commercial and research operating systems have been designed utilising the variety of operating system architectures stated above. Kernel based architectures lead to systems that provide complete operating system functionality within the kernel code itself. Examples of systems using the kernel based architecture include: Sprite [Ousterhout et al. 88] and Amoeba [Tanenbaum and van Renesse 85]; whilst examples of systems using the microkernel based architecture include: Mach 3.0 [Accetta et al. 86] and QNX [Hildebrand 92]. In these systems, the microkernel only provides: support for scheduling, virtual memory, and cross-address space IPC as in the case of Mach; and process scheduling and interprocess communication, low-level network communication, and interrupt dispatching in the case of QNX. The remaining higher level services are implemented in user level servers. However presently, there is no standard framework for a microkernel architecture, and as such, the extent and levels of the supporting functions required is still an open topic.

A microkernel allows a flexible operating system to be constructed. The following section presents the concept of a microkernel, the advantages we gain from implementing our operating system on a microkernel based architecture and finally the components found in the RHODOS microkernel called the Nucleus, one of the first originally designed and implemented microkernels.

3.1 Concept of a Microkernel

The main distinguishing characteristic of a microkernel architecture is to provide only the minimum amount of functionality and services required to support processes, message handling and an interface to the hardware. The remainder of functions and services required for the operating system to perform correctly are provided by a set of independent kernel server and system server processes. A careful balance is required when deciding at what level functions and services are to be provided in such a microkernel based operating system.

Only the functionality absolutely crucial to performance, flexibility and modularity should be located within the microkernel. If the microkernel is overloaded with functions, that could be provided by server processes, then the benefits obtained from a microkernel architecture are degraded. Conversely, if the system server processes are required to perform too many functions and operations without the support of the operating system, this would also degrade performance as duplication of functions and services would be necessary within each of those servers.

3.2 Advantages of a Microkernel Architecture

There are many benefits gained from implementing operating system based on a microkernel architecture. Using a microkernel design allows all machine dependent code to be located in the one unit. Usually written in assembly, this section is the only portion of code that should need rewriting as remaining code would only require recompilation on the new architecture. This makes the porting of an operating system to a different hardware architecture much less complicated and time consuming.

A microkernel provides an ‘abstract machine’ to the remainder of the operating system, thus hiding hardware peculiarities and also maintaining a consistent interface over varying hardware platforms. The result of this is a simplified programing base and a simplified environment to work within.

The microkernel provides a ‘virtual machine’ which can be presented to the higher layer processes in a well defined fashion, enabling an operating system to be specified in a generalized and structured way.

Maintaining the requirement of a microkernel architecture that only the minimum of functionality should be provided, forces only the mechanisms be placed into the microkernel code. While all policy and paradigm relevant code can be located within the kernel servers. This simplifies both the design of the microkernel and also system servers.

Higher performance and efficiency can be obtained from a microkernel as the total amount of code is small enough that hand tuning and optimization can be performed. Also streamlining of some sections can also improve the speed of operation. Such methods become extremely difficult as the size of the kernel increases, so a microkernel is well suited to hand tuning and modifications.

Expandability and scalability are two important factors that are required from modern operating systems. As a microkernel architecture is based on a modular system, kernel servers can be added or removed as required, enabling a system to scaled according to the situation the system is supposed to perform in. Servers can also be changed dynamically without rebuilding the entire kernel, so adding extra functions or services only requires the addition of extra servers or the updating of current servers, which can be performed whilst the system is in operating. Another advantage gained from the inherent modularity of a microkernel based system, is the ability to configure the operating system to suit the desired characteristics required by the user, thus providing a greater degree of flexibility.

As a microkernel is only required to present a minimal abstract machine to the system, this enables the execution of a diverse range of operating systems implemented as application processes. Thus, it is feasible that on a the same machine, both UNIX, MS-DOS and VMS may be run as user applications allowing different users the environments they prefer without purchasing dedicated and separate hardware.

Process migration requires several process, memory, inter process communication and/or file management services to be performed. These services are traditionally offered either entirely by a monolithic kernel or by an added module sitting on top of an operating system. However, as process migration takes a significant amount of time, the whole monolithic kernel will be ‘tied up’ performing process migration to the exclusion of all other services. Whereas in a micro-kernel, server processes provide these services in a concurrent manner. Thus, enabling the multi-tasking of process migration without a detrimental monopolisation of the processor and/or its resources.

The final benefit obtained from a microkernel architecture is the simplification of the now necessary ability to prove that the operating system will perform correctly under all circumstances and also that it will perform securely. This is another benefit from the modular structure of the microkernel and the separation of the mechanisms from the policies. The ability to prove correctness and security is difficult to obtain from monolithic or kernel based architectures due to their size and complex structure.

The most controversial decision issue leading to these two architectural concepts is networking performance. Network protocols can run in kernel space or user space. Kernel-based systems include these protocols in the kernel for performance reasons; supporters of these architecture argue that this placement reduces both latency and data copying. It was shown in [Maeda and Bershad 92] that the structure of a protocol implementation and not the location of the protocol is the primary determinant of performance.

3.3 Components of the RHODOS Microkernel

The RHODOS Nucleus has been designed to provide four primary services for the processes it supports. These services include Local Inter Process Communication (IPC), Process Dispatching, general Data and Statistic Collection, as well as support for Interrupt and Exception Handling. The position of these components is shown in Figure 2.

To achieve these four services the Nucleus is required to provide additional support in the form of Port Control for local IPC and Page Mapping which is used in the Process Dispatching. All remaining services are provided by a set of Kernel Servers.

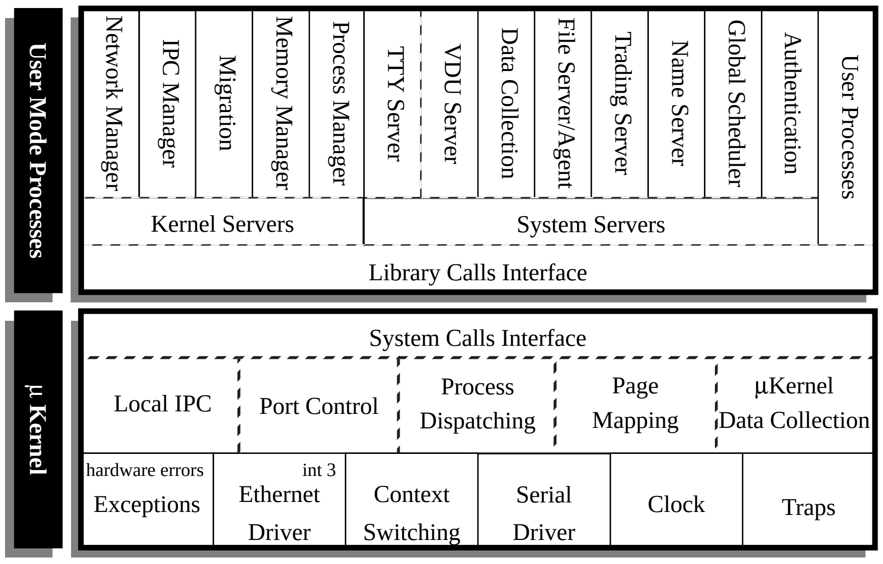

Figure 2: RHODOS Architecture

3.3.1 Local Inter Process Communication

With the exception of the shared data space of teams of medium weight processes, all RHODOS process communication is completed via message passing and remote procedure calls (RPCs); thus it is vital that an efficient and fast communication system be provided.

RHODOS provides two fundamental message passing primitives — **send** and **recv**. These primitives implement the Inter Process Communication (IPC), which is complete in the sense that it can accomplish any sort of blocking or non-blocking communication [Goscinski et al. 94]. Send and receive are combined within another system call to form the third and final IPC primitive, **call**, which is used for remote procedure calls (RPC). Receipt and acknowledgment of a call function is simply performed by a receive and send pair issued by the process providing the service. The **call** function **send**s a request to a service provider then blocks on a **recv** waiting for the result. The service provider **recv**s and processes the request supplying the result back to the origin of the message via the **send**, which is woken up on its receipt.

When the communicating processes are located on the same host, the communication service is provided by the Nucleus. All local IPC is guaranteed to be 100% reliable. If the destination of a message is not located on the local machine then the destination is assumed by the Nucleus to be located on a remote machine and requests the services of the Inter Process Communication Manager to deliver the message to the correct destination.

3.3.2 Process Dispatching

As the Nucleus is required to dispatch processes, it must maintain the states of each process (e.g., Program Counter, Registers, etc.) to enable multitasking. Once a process is to be dispatched the address space for that process must be constructed, this is achieved through the mapping of pages into a virtual address space. The details of a process including the state, and space mappings are located in a data structure known as a Process Control Block. In the 1993 version of RHODOS this is statically allocated, but the 1994 version has the ability to dynamically allocate the number of structures used within the Nucleus.

The low level scheduling of processes is also provided within the Nucleus. The scheduler used is a priority based, round robin system, with the ability to preempt process execution (i.e., event based) whilst in user mode (since this is a single threaded microkernel).

3.3.3 Interrupt Handling

The Nucleus provides an interface between an interrupt and a series of modules known as interrupt handlers. These interrupt handlers can be created and killed dynamically. Thus, enabling RHODOS to add and remove various interrupt handling routines.

An interrupt handler forms the lower level of the ‘Device Driver’ and ‘Interrupt Handler’ pair. Although not part of the Nucleus, it is the Nucleus job to attach and detach interrupt handlers from related Device Drivers[1]. Interrupt handlers are short sections of code concerned with the servicing of a computers hardware. Examples of interrupts used within the current version include two for maintaining the system clock and for scheduling (levels 7 and 5, respectively), and for other devices including the serial line, ethernet card and the SCSI device.

3.3.4 Data Collection

The Nucleus provides low level data collection on a per process basis; this data is used for forming the basis of performance measurements and gathering of statistics which can be used to tune system parameters. The data collected allows an accurate picture of the load on a given host,

1. Device Drivers are a special type of Kernel Server process that have a close association with a certain device. They provide a standard interface to user processes, whilst ensuring data from/to a device is not lost.

thus the data is also used to enable global scheduling to balance the load of the entire system. The data gathered, for instance, includes data relating to the time spent in system mode (while performing system calls); the time spent in user mode (performing normal user work or tasks); all communication operations for a process (providing details on the number of messages this process has sent/received and the total number of bytes transferred). These are logged and then retrieved when required.

4 Kernel Server Processes

As mentioned earlier, processes that have a close interaction with the Nucleus are called Kernel Servers. These servers provide the functionality that is usually placed in a monolithic kernel and thus, are trusted entities that utilise a set of privileged system calls to manipulate kernel data.

The RHODOS kernel managers are responsible for the management of the basic logical resources of a distributed system: processes and processor time, remote communication, memory, devices and network delivery protocols.

Though each kernel manager is responsible and performs specific functions, they interact following the master - slave model. The process manager is a master of the remaining managers, which do not interact with each other. This model of interaction has been selected to allow the process manager to control allocation of resources to each new and existing process.

The Kernel Servers include the Interprocess Communication Manager, Network Manager, Process Manager, Memory Manager, Migration Manager, Data Collection Manager, and Device Manager. There is also a set of device drivers for such hardware entities as the ethernet and serial devices.

4.1 Remote Communication

Since in a distributed operating system, processes may be dispersed over a large network, there needs to be a mechanism provided to support transparent remote interprocess communication. In RHODOS, this service is provided by the IPC Manager and the Network Manager.

4.1.1 Inter Process Communications Manager

Inter Process Communication between processes on the same host computer is provided by the Nucleus (as detailed in Section 3.3.1) if the destination for the message is not found by the Nucleus it is assumed that it is resides on a remote host and is forwarded onto the IPC Manager for processing. Therefore the primary role of the IPC Manager is providing destination address resolution. In terms of the ISO reference model, the IPC Manager can be seen to provide some services of the session layer. It uses the services of the transport layer and below to achieve remote communication.

The IPC Manager can be considered as the provider of the policies for which remote communication is to be achieved. The IPC Manager is also concerned with supporting message multicasting; this is achieved with the cooperation of the Name Server by assigning a single endpoint to a group of endpoints. Another service provided by the IPC Manager is with remote server liveness checking, this is required when a process is issuing remote procedure calls to servers on remote machine. A combination time-outs and remote testing is used to ensure remote calls are not stalled when a server crashes or is halted.

The final requirement of the IPC Manager is to support the Migration Manager during process migration. The IPC Manager is required to maintain transparent message passing during the migration procedure (since transparency is a primary objective of distributed operating systems). Since a process can exist simultaneously on two hosts during migration, care and cooperation is required such that messages destined for the migrating process are not lost and that their order is preserved. Thus, the IPC Manager is required to forward messages and ensure the correct order of these messages onto the process' port on the destination machine.

4.1.2 Network Manager

As policy decisions are located within the IPC Manager, the mechanisms required to implement these decisions are positioned within the Network Manager. The Network Manager utilises a transport protocol designed specifically for distributed systems; this transport protocol is known as the 'RHODOS Reliable Datagram Protocol (RRDP)' [Goscinski and Zhu 90] [Goscinski and Toomey 92].

Mechanisms which are provided by the RRDP allow a process to specify the need for reliable or unreliable delivery of a message. This is accomplished by providing three qualities of service — *at least once, exactly once* and *at most once* — the first two of which are associated with a blocking send, the remaining service indicates a non-blocking send. The semantics of these primitives, i.e., the quality of service, synchronous IPC, direct and multicast addressing are described by the parameters of the operation.

4.2 Process Manager

The job of the RHODOS Process Manager is to manage the processes that are created in RHODOS. The process manager manipulates the process queues (ready, running, timeout, blocked, frozen, zombie) and deals with parent processes waiting for child processes to exit. It provides seven functions: create a process from a file (similar to the fork() and exec() combination in UNIX); create a twin of a process either a heavy-weight or medium-weight (similar to fork() in Unix); wait for a child to exit; exit a process; kill a running process; put a process on the frozen queue prior to migration; and it handles when a process causes an exception (Segmentation Fault, Bus Fault etc.).

Users are the entities who utilise resources, and as each user is associated with one or more processes, it is these processes that are the active entities in the system. This implies that process management is an integral part of the operating system. Thus, the Process Manager is the largest kernel server, as it has to control the interaction between all the kernel servers to manage all the resources of the system. All requests for process management are sent to the process manager. These requests are processed and if resources are needed (that the process manager does not control, e.g., files, memory, devices etc.) then the Process Manager requests these resources from the appropriate kernel manager.

4.3 Space Manager

Space or memory management is closely related to hardware. However, one of the goals of RHODOS is portability across hardware platforms. Thus, RHODOS memory management has been separated into two sections: hardware dependant and hardware independent. The small hardware dependant section is found in the microkernel and the larger hardware independent section comprises RHODOS' Space Manager [Wickham, et al., 94a]. As RHODOS' Space Manager provides hardware independent functions, thus it is totally portable.

The RHODOS Space Manager performs all the management and control of memory spaces in RHODOS [Toomey 90]. A space in RHODOS is a virtual mapping of memory to a physical store (physical memory, file, etc.). As was mentioned earlier, each process is comprised of three spaces: Text, Data and Stack spaces.

The Space Manager provides the following functions: create a new space and link it to a physical store; alter the size of a specified space; create a new space which uses a copy-on-write technique to duplicate the space; allow two spaces to map to one physical store; and remove an existing space.

4.4 Migration Manager

RHODOS' Process Migration Manager has been designed to research different issues and design strategies that affect process migration. Four major areas of importance to migrating a process have been identified. These four areas are: when to suspend a process (relative to the request to migrate it), how to copy the address space of the process, when to suspend the reception of incoming messages (relative to process suspension) and how to communicate with the process after it has been migrated. Each of these four areas have had many strategies developed and implemented, and whilst claims of one way being faster than another, there has been no definitive study performed to ascertain under which conditions each strategy performs in the best manner.

Instead of deciding that a particular strategy is best for the RHODOS system, a multiple strategy process migration manager is being developed and will be used. The design of such a manager allows each of the above mentioned strategies to be used without any need to recompile. Thus, enabling the comparison of each strategy to determine which strategy will achieve the best performance for process migration in RHODOS under varying conditions.

Performance, however, is not the only important issue within the field of process migration. Reliability and Security are also of prime importance. RHODOS' Process Migration Manager utilises a transaction based approach to migration to ensure reliability and uses the services provided by the RHODOS Authentication Server to ensure secure process migration.

4.5 Data Collection Manager

The RHODOS Data Collection System will provide the functionality required to enable complete analysis during operation. The task of collection and storing the statistical information will not impact upon the ability to collect the information, because the actual task of collecting and storing information is itself a task to the operating system. However it is not known how significant the decrease in performance will be with the inclusion of a data collection system.

RHODOS' Data Collection Manager provides the ability to enable and disable data collection on a per process basis. This means that a single process can be monitored by the operating system, instead of having a single flag that will enable or disable collection from the entire operating system.

The Data Collection Manager: selects the frequency and mode of data collection, collects computational and communication patterns, supports collected statistics, provides requested data to the Global Scheduler, and provides complete process accountability through the entire operating system.

In particular, the RHODOS Data Collection Manager will collect information on the following two aspects of process performance within the operating system. Firstly, on interaction of processes with the microkernel: system calls into the microkernel, the amount of CPU time devoted to a process in user mode and system mode, statistics on local interprocess communication: usage and traffic. Secondly on interaction of processes with the Servers: Remote Procedure Call's to officially sanctioned servers (Requested call, amount of time spent processing the call etc.)

Note that the Data Collection Manager collects process statistics direct from: the microkernel, kernel and system servers and user processes. All this data yields an accurate picture of the state of the host, and is utilised by the Global Scheduler.

4.6 Device Manager

Transparency is an important feature of RHODOS, this not only includes interprocess communication between remote hosts, but also a transparent unified interface of physical devices such as serial ports, keyboards, video screens and disks. Device Drivers provide this interface. As opposed to traditional operating systems where Device Drivers are located within the kernel code, Device Drivers in RHODOS are in there own right processes with the privilege and status of Kernel Servers. The benefits obtained from implementing Device Drivers as processes include the ability to enable and disable new drivers dynamically, as well as to use normal process debugging tools whilst the Device Driver is active.

The Device Manager is the controlling entity that allows users to access a requested physical. The Device Manager maintains the binding between which process has access to which physical devices. The Device Manager allows access to these devices by using the appropriate Device Drivers.

5 System Servers

In order to support users friendly, efficiently, and securely, the RHODOS distributed operating system provides services through system servers. Each system server is in itself a testbed for the study and comparison of various distributed system mechanisms and policies. The following RHODOS system servers have been developed:

- Name Server - supports attributed naming and provides evaluation of user names onto system names;
- Trading Server - provides user autonomy and object sharing between homogeneous and heterogeneous systems through object export, import, and withdrawal;
- File Server - provides disk, file, transaction and replication services and storage for workstations without their own extensive storage;

- Global Scheduler - improves computational performance through sharing idle or lightly loaded workstations, by employing load balancing and static allocation;
- Authentication Server - authenticates users and servers by supporting one-way, two-way and conference authentication.

5.1 Name Server

The RHODOS name server is responsible for identifying all objects known to the system, and for providing the mechanism for users and/or processes to access them [Goscinski and Indulska 92].

RHODOS supports a three-level name structure with appropriate functions for mapping a name of some level to a name of an adjacent level. These levels contain: user name, denoted UName, system name, denoted SName, and location-oriented name. The RHODOS name server maps user names onto corresponding system names.

Because an ordinary user does not want to remember or simply does not know the well-defined names of an object and can only provide an imprecise description using some properties of an object, descriptive naming should be used at the user name level. Thus, the development of a naming facility for RHODOS is based on the use of *attribute names* also called *descriptive names* and follows an *attribute-based naming* approach. This approach allows both a user and a process to identify an object using a set of attributes that describe the object. Moreover, three different naming services are provided by the name server:

- conventional service — maps well-defined user names onto system names;
- enquiry service — allows a user to identify objects using imprecise descriptions and also to look up information about an object;
- selection service — identifies imprecisely-described objects and maps it onto the system name of either the object or a server which is able to provide the service, or which manages the requested object.

In addition to the naming services, the RHODOS name server allows both the provision of autonomy to individual users in an open distributed environment and the quality and efficiency of a user’s work to be improved by providing the trading service in the user level and the distributed system level [Goscinski and Ni 93]. This is achieved by exploiting the concept of the naming domain, as well as by introducing the operations of object export, import and withdrawal.

The smallest naming domain is the user domain, where users can make some of their objects invisible to other users. This solution provides a high degree of autonomy of individual users. The administrator can also make some of system objects invisible to other system, in this case, autonomy is provided at the system level. This is achieved by providing basic primitives which allow the following operations to be carried out:

- *object export* — the owner of an object may make a given object visible to users in other domains by exporting it to the specified domain(s);
- *object import* — a remote user may make an exported object accessible by importing it from the object owner’s domain, providing this object has been exported to the importer’s domain;
- *object withdrawal* — the owner of an exported object may withdrawal the object from the target domain, if he/she does not want the object being exported an more.

5.2 File Server

The RHODOS File Server uses separate strategies for the storage of file data and for the data structures required for file management [Panadiwal and Goscinski 94]. This alleviates the typical problem of loss in performance due to disk I/O in an efficient manner. Furthermore, to make the design very reliable, stable storage is provided. The design of the disk service allows the contents of a file to be distributed among more than one disk drive. Thus, for all practical purposes the design imposes virtually no restrictions on the file size and disk capacity.

Striving for reliability has also generated a new file facility architecture where a separate interface is provided for the transaction service. This service offers primitives to deal with files in a transaction mode. In order for the RHODOS File Server to match the requirements of a user the design provides three levels of locking. We claim that using transaction semantics file operations in not only database applications but also in system programming can be made resilient against system and media failure.

Due to performance, reliability, and distribution reasons the following additional issues and services have also been considered for RHODOS File Server: redirection of standard input and output, process migration, file replication, naming of distributed objects, and the placement and location of files in a distributed file facility. The RHODOS File Server’s design follows a layered approach and provides a clean interface at each level. In order to further improve performance the design of the caching module takes into consideration all the aspects of basic file and transaction services.

5.3 Global Scheduler

An important feature and design decision taken with RHODOS (as described in Section 2.1 and Section 2.2) is the ability to fully utilise the entire resources of a distributed system. One method employed by RHODOS in achieving an optimised overall system load under all situations is with the use of Global Scheduling.

Global Scheduling is concerned with making the load of all hosts within a distributed system equal, therefore improving the overall performance and throughput of processes. A combination of dynamic load balancing [Zhu and Goscinski 90] and static allocation techniques are employed to actively and passively move processes between various host CPUs. Transparency of communication and resources (a fundamental design characteristic of RHODOS) provide the freedom for processes to be migrated away from the host on which they were created to a more suitable (less loaded) host, whilst also providing for the creation of a process on a remote host. The combination of both methods of static allocation and dynamic load balancing allow for the system to react to large fluctuations in system load (using dynamic load balancing) and also to avert the case when system load remain steadily high (static allocation).

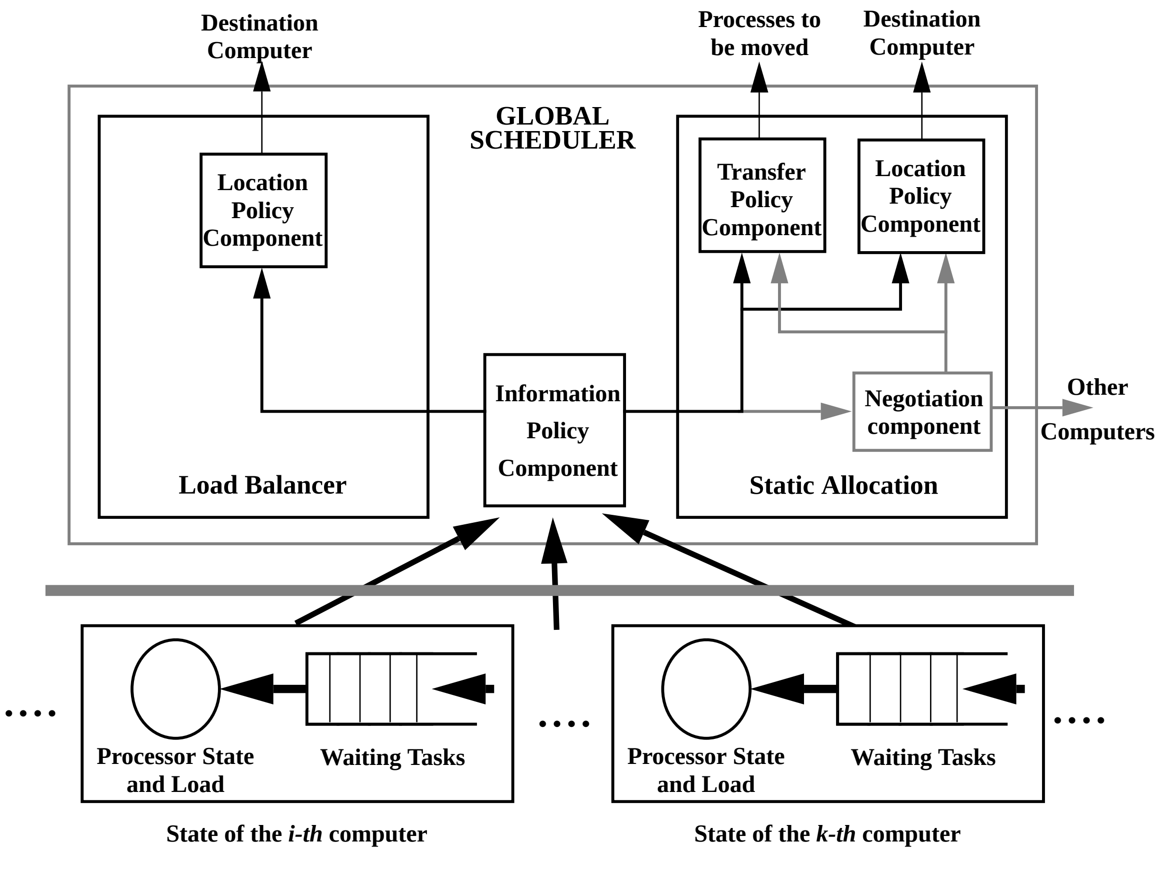

Figure 3: Components of the Global Scheduler

The Global Scheduler within RHODOS can be seen as a high level scheduler, taking into account the load details on remote hosts. With the information gathered from the distributed system, it can negotiate with Global Schedulers on remote hosts for processes to be accepted or denied. Policy and mechanism components are separated within the Global Scheduler to allow research on the differing effects each module will have on the overall system. The relationship of each of the components and their interaction with the remainder of the system are both shown in Figure 3.

5.4 Authentication Server

The RHODOS Authentication server provides basic authentication tasks including login authentication, one and two way authentication and conference authentication [Wang and Goscinski 92]. This service is based on the Identity Based Conference Key Distribution System (ICKDS), proposed by Koyama-Ohta.

The service has one central trusted authentication server and a set of authentication agents local to each workstation. When a user logs in, the central server is consulted using a zero-knowledge protocol to confirm that the users password is correct. If the user has typed in the correct password, then the central authentication server distributes the cryptographic parameters and a unique user signature to the local authentication agent, rather than to users who are not trusted.

These parameters are used by the agents to perform one-way, two-way and conference authentication, without any need to consult the central authentication server. Thus, this security scheme is well suited to a distributed operating system.

Conference authentication is unfortunately a time consuming action [De Paoli and Goscinski 93]. For this reason RHODOS' authentication server provides two versions of conference authentication. The first method of conference authentication places the participants into a star formation. The second method places the participants into a ring formation. The first method (star based) is faster than the second method (ring based) however, it is not as secure.

6 Conclusion

The main goals and design assumptions for RHODOS have been strongly influenced by a detailed study of a number of existing experimental distributed systems [Almes et al. 85], [Cheriton 88], [Finkel et al. 86], [Popek and Walker 85], [Tanenbaum and van Renesse 85], [Tavenian and Rashid 87], [Rozer et al. 88] presented in [Hildebrand 92]. They are as follows:

- Existing distributed operating systems only cover such areas as IPC, naming and protection.

This is not sufficient to deal with other problems which arise in distributed systems;

- It is very difficult to compare the existing divergent proposals in the areas of IPC, naming, and protection;
- Contradictory results published on load balancing are based on simulation, and on existing systems which have been modified by adding a Process Migration facility after the completion of the base system.

Though our system has not been fully implemented and tested, we think that the initial results are worth presenting at this point because of the ideas and approaches it contains, which are not found in other systems. In addition, we want to develop a distributed operating system which does not concentrate solely on IPC, naming, and file system, but also on other design issues which are important to distributed environments.

We have already learned the following lessons from our work:

- The provision of a load balancing facility requires a very efficient and effective process migration facility;
- The Process Migration facility must be treated as intrinsic part of the Kernel —it cannot be designed and developed after completion of the Kernel. It needs to cooperate very closely with IPC Manager, Memory Manager and the Nucleus;
- An effective naming system should be developed based on attributed names, to provide access to an object based on the object attributes not only names. The attributed naming may provide a wide range of services from information about objects to a selection of an object with matching attributes.
- Clear division between User Names and System Names allows the study of different distribution structures;
- A careful integrated design greatly simplifies the study of different protection schemes;
- Load balancing for a “local” distributed system should be based on a distributed approach. Allocation of other resources can be performed using a centralized approach;
- Construction using a highly-modular fashion, with hooks for test instrumentation, allows RHODOS to be used as a test bed for system ideas [Hobbs et al. 92]. At the same time, its modularity will enable it to be configured as a working product.

6.1 Innovative and improved ideas

A number of the ideas used in the design of RHODOS are innovative, either in their entirety, or in the manner or context in which they are used. They are as follows:

- RHODOS has a single unified model for interprocess communication using message passing which underlies *all* system communications. This includes all hardware and software interrupts;
- RHODOS is capable of supporting a variety of process types. Currently including heavy- and medium-weight processes;
- RHODOS is designed and implemented to study distributed operating systems in a generalised way, instead of focusing on specific issues such as communication performance or naming. It can be used to study:
    - IPC primitives and their effectiveness in different environments;
    - protection systems: access lists, capabilities, or a mixture of them, as well as non-discretionary mechanisms;
    - attributed naming;
    - object (resource and service) trading;
    - different naming distribution and resolution concepts;
    - global scheduling combining dynamic load balancing algorithms and static allocation algorithms;
    - process migration policy and mechanisms;
    - communication security;
    - authentication, in particular conference authentication;
    - distributed file facility.

6.2 Current Work

The current focus of attention within RHODOS relates primarily to the interaction between the Nucleus and the set of kernel servers. Development and enhancements to the kernel servers have led to the conclusion that the current version of the system has some deficiencies. These include the inability to preemptively schedule processes whilst executing system calls; and the inability to perform paging whilst still holding to the microkernel design paradigm. Therefore the current direction of our research and development has been in the area of multi-threaded microkernels [Wickham et al. 94b]. A multi-threaded microkernel and an enhancing the memory management enables us to overcome the problems faced with the present system (single-threaded). Thus, these small changes to the structure of the Nucleus will allow the implementation of both preemptive scheduling and paging in a simple and efficient form, whilst still conforming to the microkernel paradigm.

6.3 Future Work

The RHODOS platform is evolving into an extremely valuable tool for the study of distributed systems and distributed operating systems. There are a number of research areas available for study both in the current version of the system and even more so within the multi-threaded version.

Once the implementation of the multi-threaded microkernel has been completed, this will open up new areas of research. Firstly, with the enhanced memory management paging will be available, allowing research into paging strategies to commence. With the advent of paging, process migration will be able to migrate address spaces by paging to a common file server. This allows the comparison of different strategies for address space migration to be researched. Another benefit of the enhanced memory management will enable performance measurements of the whole system (microkernel, kernel, system and user process interaction) to be undertaken, and hence, refinement and improvement of the whole system will be performed. With the timing functions in place, data will be available to the global scheduler which will give an accurate picture of the load on a host. This information can then be used to equalise the load of the whole system, by using both static allocation and load balancing.

7 References

[Accetta et al. 86] J. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevavian, M. Young. *“Mach: A New Kernel Foundation for Unix Development.”* Proceedings of the Summer 1986 USENIX Conference, July, 1986, pp. 93-113.

[Almes et al. 85] G. Almes, A. Black, E. Lazowska, J. Noe. *“The Eden system: A Technical Review”*. IEEE Transactions on Software Engineering, SE-11, 43-59.

[Cheriton 88] D. Cheriton. *“The V Distributed System”*. Communications of the ACM, 31(3), 314-33.

[De Paoli and Goscinski 93] D. De Paoli and A. Goscinski. *“Times of Cryptographic Parameter*

Generation, and Key Computation and Distribution for the Star-based and Ring-based Conference Authentication Facilities". Technical Report TR C93/14, School of Computing and Mathematics, Deakin University, Geelong, May, 1993.

[Finkel et al. 86] R. Finkel, et al. *"The Charlotte Distributed Operating System"*. Computer Science Technical Report #653, University of Wisconsin-Madison, Computer Science Department. 1986.

[Gerrity et al. 91] G. Gerrity, A. Goscinski, J. Indulska, W. Toomey and Z. Zhu. *"RHODOS — A test bed for the studying Design Issues in Distributed Operating Systems"*. Proceedings of the 2nd Singapore International Conference on Networks (SINCON'91). September 1991.

[Goscinski 91] A. Goscinski. *"Distributed Operating Systems: A Logical Design"*. Addison-Wesley, 1991.

[Goscinski et al. 94] A. Goscinski, M. Hobbs, G. Wickham, P. Joyce. *"Message Passing and RPC-based Interprocess Communication Mechanisms in the RHODOS Microkernel"*. Technical Report TR C94/, School of Computing and Mathematics, Deakin University, Geelong, (in preparation).

[Goscinski and Indulska 92] A. Goscinski and J. Indulska. *"The RHODOS Naming Facility"*. Distributed Processing Technical Committee Newsletter, invited paper, No 1.

[Goscinski and Ni 93] A. Goscinski and Y. Ni. *"Object Trading in Open Systems"*. Proceedings of the International Conference on Open Distributed Processing — ICODP'93, Berlin, September 1993.

[Goscinski and Toomey 92] A. Goscinski and W. Toomey. *"A Fast Reliable Transport Protocol for Real-Time Distributed Systems"*. Proceedings of the IEEE International Workshop on Emerging Technologies and Factory Automation EFTA's92, August 11-14, Melbourne.

[Goscinski and Zhu 90] A. Goscinski, W. Zhu. *"The Development and Performance Study of the RHODOS Reliable Datagram Protocol (RRDP)"*. Proceedings of the 10th International Conference on Computer Communication, ICCC'90, November 1990, New Delhi, India, Naroasa Publishing House.

[Hildebrand 92] D. Hildebrand. *"An Architectural Overview of QNX"*. Proceedings of Workshop on Microkernel and other Kernel Architectures. April 1992.

[Hobbs et al. 92] M. Hobbs, W. Toomey, G. Wickham. *"Booting of the RHODOS Distributed Operating System"*. Technical Report TR C92/3, School of Computing and Mathematics, Deakin University, Geelong, August 1992.

[Maeda and Bershad 92] C. Maeda and B. Bershad. *"Networking Performance for*

Microkernels”.

[Ousterhout et al. 88] J. Ousterhout, A. Cherensen, F. Douglis, M. Nelson, B. Welch. *“The Sprite Network Operating System”*. Computer, 21(2), pages 23-36.

[Popek and Walker 85] G. Popek, B. Walker. *“The LOCUS Distributed System Architecture”*. Cambridge, Mass: The MIT Press.

[Rozer et al. 88] M. Rozier, V. Abrossimov, F. Armand, M. Gien, M. Guillemont, F. Hermann and C. Kaiser. *“Overview of the Chorus Distributed Operating System”*. Montigny-le-Bretonneux (France), June 1988.

[Tanenbaum and van Renesse 85] A. Tanenbaum, R. van Renesse. *“Distributed Operating Systems”*. Computing Surveys, 17(4).

[Tavenian and Rashid 87] A. Tevanian, R. Rashid. *“Mach: A Basis for Future UNIX Development”*. Carnegie-Mellon University, Department of Computer Science.

[Toomey 90] W. Toomey. *“Memory Management in RHODOS”*. Technical Report CS90/19, Department of Computer Science, University College, University of New South Wales. May 1990.

[Panadiwal and Goscinski 94] R. Panadiwal and A. Goscinski. *“A High Performance and Reliable Distributed File Facility”*. Proceedings of the 14th International Conference on Distributed Computing Systems. Poznan, Poland, June 21-24, 1994.

[Wang and Goscinski 92] M. Wang and A. Goscinski. *“The Development and Testing of an Authentication Service for RHODOS”*. Technical Report CS90/7, Department of Computer Science, University College, University of New South Wales, September 1992.

[Wickham, et al., 94a] G. Wickham, D. De Paoli, M. Hobbs. *“The RHODOS Space Manager 1993”*. Technical Report TR C94/, School of Computing and Mathematics, Deakin University, Geelong, (in preparation).

[Wickham et al. 94b] G. Wickham, M. Hobbs, A. Goscinski. *“Research into the Development of the RHODOS Multi Threaded Microkernel”*. Technical Report TR C94/, School of Computing and Mathematics, Deakin University, Geelong, (in preparation).

[Zhu and Goscinski 90] W. Zhu and A. Goscinski. *“The Development of a Load Balancing Server and Process Migration Manager for RHODOS”*. Technical Report CS90/47, Department of Computer Science, University College, University of New South Wales, June 1990.