SE Computational Systems, SS 2004, University of Salzburg

Xen - The Art of Virtualization

Paul T. Barham, Boris Dragovic, Keir Fraser,
Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer

July 7, 2004

Author:
Marcus Harringer.
mharring@cosy.sbg.ac.at

Academic Supervisor:
Univ.-Prof.Dr. Christoph Kirsch,
ck@cs.uni-salzburg.at

Abstract

This article is a survey about the paper *Xen and the Art of Virtualization* [1], which was presented at the SOSP 2003. Xen is a virtual machine monitor that enables a machine to be partitioned into several virtual machines. Each of these virtual machines is able to run an operating system. Xen tries to minimize the overhead by using partial virtualization, also called *paravirtualization*. Moreover Xen provides performance isolation by using an idealized virtual machine abstraction. To use a guest operating system, the kernel must be *ported* in order to use the virtual machine.

Contents

1 Introduction

There are a lot of different approaches in virtualizing resources of a modern computer. Some make use of specialized hardware or cannot support general purpose operating systems. Others use full virtualization on an x86 machine at the expense of performance. Moreover performance isolation is not an issue at all. Xen tries to overcome these problems by providing a high performance virtual machine monitor with performance isolation. It is important to note that there is a huge difference between a virtual machine and a *virtual machine monitor*. A virtual machine needs a host operating system whereas a virtual machine monitor runs on bare metal. Clearly, the monitor design can improve the performance significantly in comparison to a conventional virtual machine.

To sum up, the major challenges in building a virtual machine are:

- **Performance isolation:** Running applications in one guestOS must not affect the performance of another guestOS. Important resources are: CPU time, memory, and disk bandwidth.
- **Support for a variety of operating systems:** It is important to support general purpose operating systems. Otherwise the majority of potential users won't use the virtual machine.
- **Small performance overhead:** The virtualization layer should multiplex resources as fast as possible.

As already said, there is one drawback when using Xen: The guest operating system has to be modified. Currently there is a port of Linux-2.4, called XenoLinux, available. The Xen virtual machine monitor is used in the XenoServer project [3] . There, it is possible for client to rent an instance of a guestOS and to install and run several servers, completely isolated from other servers, administrated by other clients, but running on the same machine, but within another guestOS. The motivation that they have seems to be focused on reducing the complexity of webserver setup and maintainance.

In order to achieve performance isolation, Xen uses a similar approach as in the Exokernel [4] project. Xen multiplexes hardware resources at the granularity of an entire operating system. This approach also eliminates the problem of QoS crosstalk: It is nearly impossible to account all resource usage of a process. Consider for example complex page replacement algorithms.

This paper is structured as follows: (section 2) gives an overview of the virtual machine, (section 3) explains design details, (section 4) discusses several evaluations, and (section 5) sums up the problems and challenges in building a virtual machine, and the design and implementation of Xen.

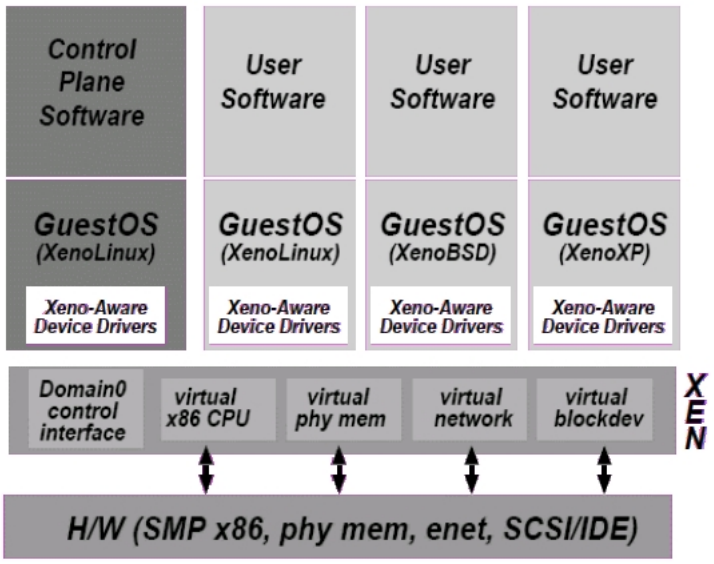

Figure 1: The structure of a machine running the Xen virtual machine monitor. Source [1].

2 Overview

Figure 1 shows the overall architecture of Xen. Xen is running on the bare machine and therefore has the same basic functionality as an operating system. Moreover, Xen provides a virtualization mechanism for guest operating systems. There is one special guest operating system in Xen. This special guestOS is used to administrate the virtual machine monitor, for example adding and removing operating systems, setting resource usage limits and so on.

2.1 Design Principles

There are two fundamental design issues: Full virtualization and Paravirtualization. One big advantage of full virtualization is that you can run unmodified operating systems. But there are a number of drawbacks:
The x86 architecture implies several problems. First, certain supervisor instructions fail silently instead of causing a trap. Since every instruction must be caught by the VMM, this can cause serious problems. Secondly, it is very difficult to virtualize the x86 MMU. VMWare's ESX Server is a virtual machine monitor that solves these problems, but at the cost of performance. The ESX Server dynamically inserts traps into the hosted machine code and uses shadow page tables to virtualize the MMU. This approach is very costly if the guest operating system continuously updates its pages. To bypass these problems, Xen VMM uses *paravirtualization*. This approach is more effective but the guest operating system has to be modified. By using paravirtualization, each guest operating system is responsible for performing its own paging mechanism. This idea is not new. The Nemesis [5] operating system uses this kind of user-level paging, called self-paging. The

next sections will describe the design in more detail.

2.2 The Virtual Machine Interface

This section presents the internal design of the paravirtualized subsystems: Memory Management, CPU, and Device I/O.

2.2.1 Memory Management

Virtualizing memory is a quite challenging task. Again, there are several problems with the x86 architecture. The biggest disadvantage is that the x86 CPU does not have a software-managed translation lookaside buffer (TLB). A software-managed TLB could easily be virtualized. A tagged TLB would be another useful feature. By tagging entries in the TLB, the hypervisor (VMM) and several guest operating systems can reside in memory without being flushed out at each context switch. However, the x86 CPU neither supports a software managed TLB nor a tagged TLB. This means that TLB misses are serviced automatically and that every context switch requires a complete TLB flush. Xen solves these problems in several ways. TLB flushes are avoided by letting the virtual machine monitor reside at the top of every address space. To avoid the problems implied by the hardware-managed TLB, the developers of Xen made the decision that each guestOS is responsible for allocating and managing the hardware page tables. Imagine, for example, the following situation:
A guestOS requires a new page table. First, the guestOS allocates pages from its own memory reservation and registers it with Xen. At that time, each update to the page table, has to be validated by Xen. As a matter of performance, it is possible to *batch* update requests, in order to limit the number of hypervisor calls. Virtualizing segments is done in a similar fashion.

2.2.2 CPU

The concepts of protection levels and call gates were pioneered in MULTICS, where they were viewed as protection rings. At level 0, we find the kernel of the operating system, which handles I/O, memory management and other critical matters [6]. At level 3 we find the user applications, which is the least privileged ring. In Xen, we have three different levels of protection:

1. Hypervisor Level.
2. Operating System Level.
3. Application Level.

Normally, an operating system will run in ring 0, which is the most privileged level. In Xen, the virtual machine monitor runs in ring 0. Guest operating systems have to be modified to run in ring 1. Another possibility would be, that the operating system runs as

the highest priority process in user space. But the concept of using three rings works fine on a x86 CPU since there is support for four different protection levels (rings). Every time a guest operating system calls a privileged instruction, a trap occurs and control is given to the hypervisor, the virtual machine monitor. The mechanism works the same way as in a system without an additional protection layer.
Special care has to be taken with the page fault handler. Normally the page fault handler would read the faulting address from a privileged register. As said before, an operating system in Xen runs in ring 1 and does not have privileges to access this register. Hence, Xen copies the exception stack frame on the stack of the guestOS and returns control to the appropriate handler. For all other exceptions it is possible to register a so called *fast exception handler* with Xen. This exception handler is directly accessable by the processor, without indirecting through Xen.

But what about safety ? Each handler is validated if the handler's code does not specify execution in ring 0. A problem occurs if the handler itself is not paged into memory. In this case the operating system is simply terminated.

2.2.3 Device I/O

Xen provides a simple and clean set of device abstractions. I/O data is transferred via *asynchronous buffer descriptor rings.* These rings allow efficient data transfer and validation. Moreover Xen supports an asynchronous light-weight event delivery mechanism. Control- and Datatransfer will be discussed in detail in the next chapter.

2.3 Control Management

In Xen, a machine is subdivided into domains. *Domain0* runs the control plane software, while all other domains contain the guest operating systems (see figure 1). The control software runs within a guest operating system on the top of the virtual layer. At boot-time only the operating system in *Domain0* is started. This initial domain hosts the application-level management software. With this software you can create new domains or terminate old ones, set scheduling parameters, physical memory allocation, or handle access to devices.

2.4 The Cost of Porting an OS to Xen

As already said, there is one drawback. A guest operating system has to be modified in order to run on the virtual machine monitor. A question that arises now is, *what are the costs of porting an operating system to Xen ?.* Table 1 gives an overview of the lines of code that have to be ported. Note, that the port of Windows XP and NetBSD is not yet complete.

OS subsection	Linux	Windows XP
Architecture independent	78	1299
Virtual network driver	484	-
Virtual block-device driver	1070	3321
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
Portion of total x86 code base	1,36 %	0.04 %

Table 1: Lines of code for porting an OS to Xen. Source [1].

3 Detailed Design

3.1 Control Transfer

Basically there are two kinds of control interactions. Control, given from Xen to an operating system and control given from an operating system to Xen. Xen supports this two different mechanisms.

1. Synchronous hypercalls.
2. Asynchronous event delivery mechanism.

Synchronous hypercalls include all kinds of privileged instructions like page-table updates. A hypercall is a synonym for a systemcall. The asynchronous event delivery mechanism is used to forward device interrupts to domains. For example, an interrupt occured, indicating that a new packet has been received at the network interface card. Because of the asynchronous mechanism, events that are not serviced immediately have to be stored. Remember the event semantics, that no event is allowed to be lost or ignored. Therefore, Xen stores pending events in a bitmask for each domain. Disabling interrupts by an operating system, because of atomic instructions for example, can be achieved by setting a special flag. This is, from the viewpoint of a guestOS, like disabling interrupts on a real processor.

3.2 Data Transfer

Having an additional layer between the application and the device means additional overhead. Especially, when transferring data from and to a domain the indirection through Xen should be as short as possible. In Xen they decided to use asynchronous buffer rings for event *and* data delivery. Such buffer rings provide an efficient mechanism to transfer data. The ring is a circular queue that does not contain the data itself. Data can be accessed via descriptors. Figure 2 shows such a ring.

There are four actors in this transfer scheme:

- Request producer.

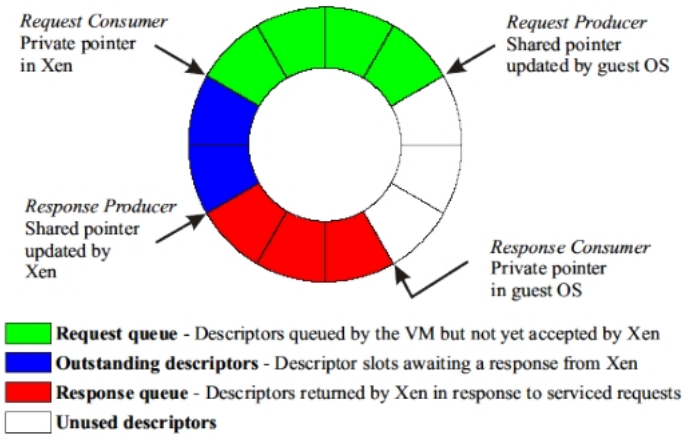

Figure 2: The structure of asynchronous I/O rings, used for data transfer between Xen and guest OSes. Source [1].

- Request consumer.
- Response producer.
- Respone consumer.

A guestOS acts as a request producer and places requests on the ring. Xen, which is the consumer of these requests, takes them. After servicing these requests, Xen acts as a producer an places the responses on the ring. Finally, the operating system consumes these respones. There is no requirement that the requests have to be serviced in order. Each guestOS gives every request a unique ID. The idea behind this scheme is that Xen can reorder requests. For example, reading from a harddrive is more effective when reading blocks from sectors that are physically adjacent. This reduces the moving of the read head. Moreover, it is a good idea to queue some requests and to make just one hypercall.

3.3 Subsystem Virtualization

3.3.1 Domain Scheduling

Intuitively, someone would think that a simple round robin scheme would be most desirable. Round Robin is a *fair* scheduler, that gives each domain a fixed number of slices. Nevertheless, the currently implementation of Xen uses a scheduling scheme called *Borrowed Virtual Time Scheduling (BVT)* [7]. They have decided to use this scheduler because it is, firstly an universal scheduler, secondly it is work-conserving and thirdly it has a special mechanism for low-latency dispatch. This reduces the effect of virtualization.

3.3.2 Virtual Address Translation

As explained in the previous section, virtualizing memory is a very difficult task when using an x86 processor. VMWare's ESX server uses virtual page tables to solve this problem. Virtual page tables are not visible to the memory-management unit. Using this scheme significantly decreases the performance of the system. To paravirtualize the memory will mitigate this problem. In Xen, physical resources are visible to the operating system. Xen is only involved in page table updates. Therefore, Xen has to register guestOS page tables *directly* with the MMU. The virtual machine monitor has to validate requests(hypercalls) efficiently. This works as follows.
Each machine page frame is associated one of the following *types*.

- PD = Page Directory.
- PT = Page Table.
- LDT = Local Descriptor Table.
- GDT = Global Descriptor Table.
- RW = writeable.

These types are mutually exclusive. For example. a page frame can either be *writeable* or a *page table*. In this scheme it is not possible to write a page table.

3.3.3 Physical Memory

Physical memory is statically partitioned. Each domain has its fixed initial memory reservation at the time of its creation. This provides a strong isolation. At the time of creation it is possible to define a maximum allowable reservation limit. This means that if the memory preassure within a domain becomes to heavy, the guestOS can reclaim more pages from Xen. Each OS should use as less pages as possible. Pages are given and take via a balloon driver [8]. Each guestOS must support this driver. The idea of *ballooning* is taken from VMWare's ESX server. It is important to note that Xen has control over the balloon driver. Xen can inflate or deflate the balloon. Figure 3 shows the basic principle.

Inflating the balloon causes increased memory preassure within the guest operating system's memory. So the guestOS is forced to use its own memory management algorithms or even be forced to swap pages out to the virtual disk. Similarly, deflating the balloon decreases the memory preassure.

Furthermore, each guestOS is responsible for its illusion of continuous memory. This implies that mapping from the physical to hardware address is also done by the guestOS itself by using a simple indexed array. For efficient address translation, Xen's shared translation array can be used. Each guestOS can read without intervention of Xen. All updates have to be validated.

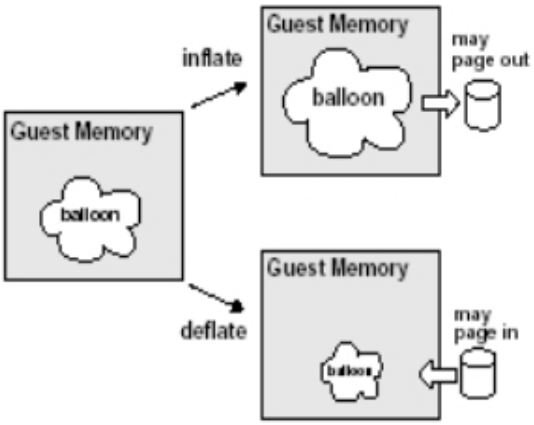

Figure 3: The idea of ballooning. Source [8].

3.3.4 Network

Each guestOS can use one or more virtual network interfaces (VIF). To transfer data between an operating system and the network, buffer rings, described earlier, are used. One buffer ring for the transmission, and one buffer ring for the receiving of packets. For example, if a guestOS wants to send a packet it enqueues a buffer descriptor on the ring. The buffer descriptor indicates the packet to send. Xen then only takes the packet header for decisions, concerning certain filtering rules. The requests from all domains are serviced in a round robin fashion, since this ensures fairness. On the other hand, if Xen receives a packet from the network, it has to copy the packet into the address space of the operating system. Copying data between address spaces is a very expensive operation. In Xen, they used a very efficient transfer mechanism. Every time when Xen wants to copy a packet to an OS it makes an exchange of pages. This means that the guestOS gives one of its free pages and gets the page frame containing the packet. Of course, this requires page-aligned receive buffers. If a guestOS has no free page to exchange then the packet is simply dropped.

3.3.5 Disk

Xen offers the abstraction of virtual block devices (VBDs). The control plane software running in *Domain0* has unlimited access to every device. With this software you can set the ownership and other access control information. Each block device is accessed via the I/O buffer descriptor ring mechanism. As network requests, disk requests are serviced in a round robin fashion. It is useful for an operating system to batch requests. First, only one hypercall is needed and second, Xen can reorder requests. It is also possible for a guestOS to insert *reorder barriers*. This forces Xen not to reorder requests. This can make sense when using a write-ahead log.

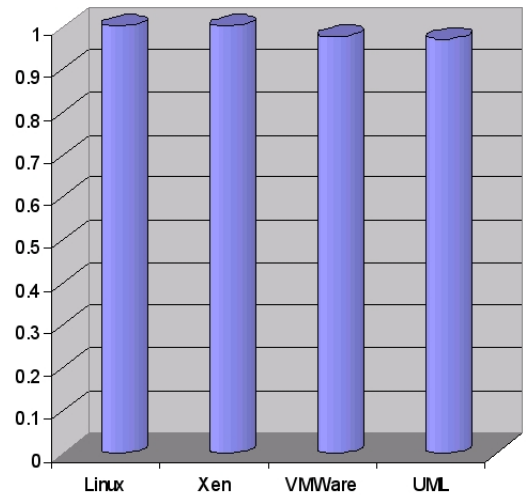

Figure 4: SPEC INT2000:Relative performance of native Linux, XenoLinux, VMWare Workstation, User Mode Linux. Source [1].

4 Evaluation

This section presents some performance evaluations. The first subsection shows the relative performance of Xen. Xen is compared to native Linux-2.4, VMWare's Workstation and User Mode Linux. The reader may wonder why the comparison excludes VMWare's ESX server. This is because the end-user license agreement dissallows the publication of quantitative performance measurements. So they used VMWare's Workstation, which is not a virtual machine monitor, meaning that this virtual machine needs a host operating system. It's clear that the workstation is therefore not as fast as the ESX server. In User-Mode Linux it is possible to run Linux within Linux.

4.1 Relative Performance

For testing Xen certain SPEC suits were used. SPEC stands for Standard Performance Evaluation Cooperation. The SPEC CPU suite includes tests of long running computationally intensive applications. Figure 4 shows the results. All systems perform about the same. This is mainly because most computation is done in userspace.

Another evaluation is made with the Open Source Database Benchmark suite. In the paper they presented the results for online transaction processing. OLTP causes many synchronous disk operations. Here we see a huge difference between Xen and Vmware and UML. Xen performs nearly as well as native Linux. This is mainly because of the paravirtualization. Results are shown in figure 5.

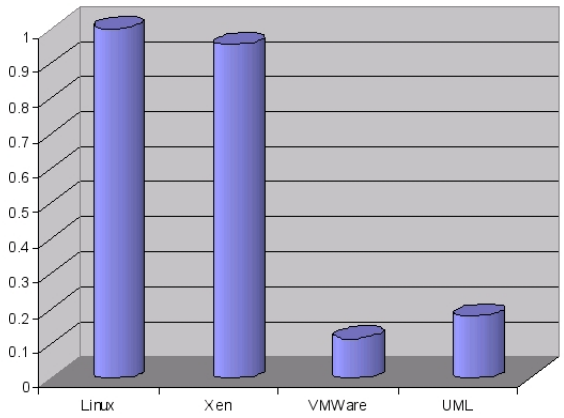

Figure 5: OSDB-OLTP (tup/s) :Relative performance of native Linux, XenoLinux, VMWare Workstation, User Mode Linux. Source [1].

4.2 Scalability

One of the goals of Xen was to be scalable up to 100 instances. Again, the SPEC INT2000 test suit was used. Figure 6 shows this.

Note, that the maximum throughput is 2.0, because of a dual processor machine. Figure 6 illustrates two different configurations: 50ms time slice and 5ms time slice. Native Linux identifies the processes as compute-bound and schedules them with a 50ms slice. If Xen also uses a 50ms timeslice it performs nearly as well as native Linux.

5 Conclusion

- Q: What problem does the paper address ?

  Virtualization of a machine and its resources, to support multiple operating systems, that can coexist at the same time. Xen tries to overcome the virtualization problems with the x86 architecture. Especially the virtualization of the memory management unit (MMU) and the hardware-managed TLB is a quite difficult task.

  Another important fact is performance.

- Q: How is it different from previous work, if any ?

  Most other approaches to virtualize a machine, like User-Mode Linux or VMWare's workstation, need an operating system that hosts the virtual machine. And on that virtual machine it is possible to install a virtual operating system. Xen itself is a small operating system, compareable to the microkernel architecture. VMWare's ESX server uses a similar approach. The main difference to all other systems is that Xen uses the concept of paravirtualization. Hence, operating systems know that they are executed on a virtual machine and not on bare metal. The operating system has

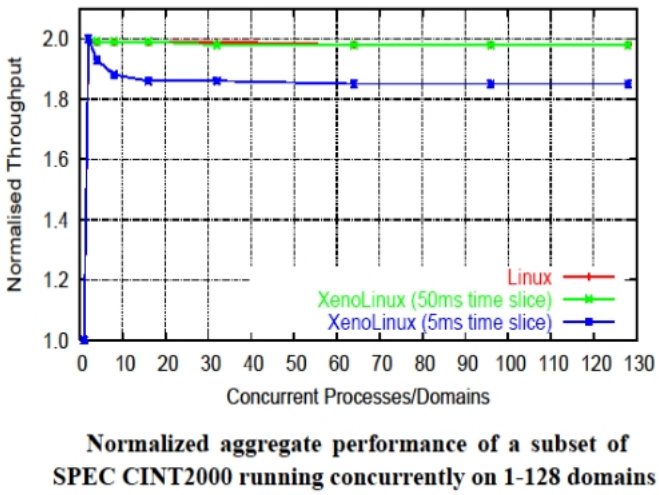

Figure 6: The structure of a machine running the Xen virtual machine monitor. Source [1].

to be ported in order to run on Xen. The advantage is that the operating system can be optimized for that special purpose.

- Q: What is the approach used to solve the problem ?

  Xen is a high performance virtual machine monitor (VMM) that supports performance isolation and up to 100 instances of operating systems. To achieve these goals, Xen uses the concept of paravirtalization. Paravirtualization means that the virtual machine is not an exact image of the underlying hardware. For efficient data and control transfer they use buffer descriptor rings. These rings do not directly contain data. Each guestOS plays the role of a request producer and response consumer. Xen services request and acts the other way around. For performance issues Xen can reorder requests. Moreover each guestOS has to perform its own paging.

  Systemcalls in Xen are called hypercalls. These are privileged instructions that must be caught by Xen. In Xen there are three different protection levels. This is not a problem since the CPU supports at least three protection levels. Interrupts for operating systems can be disabled by setting a special flag in Xen. Since all events must be serviced, Xen has to store those events until consumption. Setting this flag is, from the viewpoint of the guestOS, the same as disabling interrupts on the CPU.

  To install guestOSes and to administrate the machine, Xen uses a special purpose guestOS, running in *Domain0*. This operating system hosts the control plane soft-

ware. It is possible to set scheduling parameter, accessibility issues, memory size and so on, via this control software. At startup only this special guestOS is loaded.

- Q: How does the paper support its arguments and conclusions ?

  They have implemented the virtual machine monitor. Additionaly they ported Linux-2.4, called XenoLinux, to run on Xen. The ports of Windows XP and NetBSD are not yet complete. A lot of performance measurements proof that Xen performs really well in comparison to other virtual machines , and that it scales up to 100 guestOSes.

- Q: What are the pros and cons of Xen ?

  Pros:

  - very high speed.
  - high scalability with good resource isolation.
  - fine grained resource allocation.
  - application compatibility.
  - open source.

  Cons:

  - guestOSes have to be ported to the Xen architecture.
  - relatively large amount of codelines to patch linux kernels.
  - currently less features than UML.
  - currently only a support for maximum 4GB memory.

References

[1] Paul T. Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer "Xen and the Art of Virtualization", *SOSP 2003 , Pages 164-177.*

[2] G.Banga, P.Druschel, and J.C. Mogul "A new facility for resource management in server systems", *In Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI 1999) , pages 45-58, Feb 1999.*

[3] K.A.Fraser, S.M.Hand, T.L.Harris, I.M.Leslie, and I.A.Pratt. "The Xenoserver computing infrastructure. Technical Report UCAM-CL-TR-522", *University of Cambridge, Computer Lab.* January 2003.

[4] M.F.Kaashoek, D.R.Engler, G.R.Granger, H.M.Briceno, R.Hunt, D.Mazieres, T.Pinckney, R.Grimm, J.Jannotti, and K.Mackenzie "Application performance and

flexibility on Exokernel systems", *In Proceedings of the 16th ACM SIGOPS Symposium on Operating Systems Principles, volume 31(5) of ACM Operating Systems Review, pages 52-65* October 2003.

[5] S.Hand "Selfpaging in the Nemisis operating system", *In Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI 1999), pages73-86*, October 1999.

[6] A.S.Tanenbaum, A.S.Woodhull "Operating Systems, Design and Implementation", *PRENTICE HALL, Second Edition, 1997*

[7] K.J.Duda and D.R. Cheriton, "Borrowed Virtual Time Scheduling: support latency-sensitive threads in a general-purpose scheduler". *In Proceedings of the 17th ACM SIGOPS Symposium on Operating Systems Principles, volume 33(5) of ACM Operating Review, pages 261-276, Kiawah Island Resort, SC,USA* December 1999.

[8] C.A.Waldspurger "Memory resource management in VMWare ESX Server", *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, (OSDI 2002), ACM Operating Systems Review, Special Issue, pages 181-194, Boston, MA, USA*, December 2002.