# RAPHA: Rate-Based Page Fault Handling Mechanism in Virtual Memory System

**Young-Woong Ko*, Hyuck Yoo**

**Department of Computer Science, Korea University**

**1,5-ka, Anam-dong, Sungbuk-ku, SEOUL, 136-701, KOREA**

**E-mail: {yuko, hxy}@os.korea.ac.kr**

## *Abstract*

In virtual memory system, whenever a new task is dynamically launched, it suffers from transient overload that caused by excessive page fault handling. As transient overload state occurred, the QoS of multimedia application may be degraded because the deadline miss ratio is increased. In this paper, we present efficient overload control mechanism for virtual memory system called RAPHA(RAte-based Page fault HAndling). A significant feature of the RAPHA algorithm is resource reservation for kernel activity and page fault dispersion based on per process limit. The RAPHA algorithm is implemented in the Linux operating system and its performance measured. The results demonstrate RAPHA's superior performance in supporting multimedia applications. Experiment result shows that RAPHA mechanism could achieve 10%-20% reduction in deadline miss ratio and 50%-60% reduction in average delay.

**Keywords** : virtual memory, soft real-time, linux, page fault

## 1. Introduction

Recent advances in computing and communication technologies have led to the emergence of a wide variety of applications with diverse performance requirements. Today's general-purpose operating systems are required to support a mix of conventional best effort, throughput-intensive and soft real-time applications. Especially, soft real-time applications have time-constrained data types such as digital audio and video. Audio and video streams periodically change the values of the media data over time, and they must be presented at a specific deadline. A key of designing a multimedia system is whether media data are handled properly in terms of time. Therefore, the importance of a scheduling algorithm is well recognized, and real-time scheduling techniques[1, 2, 3, 4, 5, 6, 7] have been used. Although they are important for a multimedia system, the area of memory management has been totally ignored and is not supported any concept of Quality of Service. In this paper, we focus on transient overload state that caused by virtual memory architecture. In virtual memory system, whenever a new task is dynamically launched, virtual memory system suffers from extensive page fault that cause transient overload state. When a process attempts to access a page that is not resident in memory, the system generates page fault event and handles fault that causes unpredictable delay. So current virtual memory system is not appropriate for real-time system, because it can increase the deadline miss ratio of real-time task.

In this paper, we discuss about transient overload caused by excessive page fault handling. Furthermore, we suggest ideas that prevent transient overload. The contribution of this paper is as follows. First, we show how to control excessive page fault in current virtual memory system. Second, we consider several real-world applications and demonstrate that RAPHA enable to provide benefits such as predictable resource management. For instance, we show that RAPHA enables a multimedia player to display MPEG-1 files at their real-time rates regardless of the transient overload state. Finally, we show that the implementation overheads of resource management techniques are small, making them a practical choice for general-purpose operating systems such as Solaris[21] and Linux [22].

The rest of this paper is structured as follows. Section 2 describes related works, and section 3 presents current

virtual memory problems and discusses about transient overload state. Section 4 discusses the principles underlying the design of RAPHA and briefly describes each component employed by RAPHA. Section 5 presents the results of our experimental evaluation, and finally, Section 6 presents some concluding remarks.

## 2. Related works

The growing popularity of the multimedia application has resulted in several research efforts that have focused on the design of predictable resource management mechanisms. The problem of ensuring that applications are guaranteed a required quality of service and do not violate certain restrictions on resource usage has recently attracted a lot of attention. Consequently, several resource management techniques have been proposed for the predictable allocation of memory [8, 9, 10, 11, 12, 13, 14, 15, 16].

One way to support predictable memory management is memory-wiring technique. In traditional Unix environment, the function mlock(*addr*, *len*) causes those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes to be memory resident until unlocked or until the process exits. While memory wiring prevent real-time task from being paged out, it could not prevent transient overload when a task is dynamically created environment. Furthermore, the excessive use of mlock() tends to lead overall process to starvation. Resource reservation model is widely used in various operating systems such as real-time Mach [1], Rialto [19] and Eclipse [20]. This scheme is implemented reservation-based resource scheduling that guarantees applications a predictable resource share over periodic time interval. Although resource reservation is widely used, however, resource reservation model has been focused on CPU and disk issues. Another mechanism to support predictable memory management is external pager such as Nemesis[10], V++[11], Exokernel[15], SPIN[17] and VINO[18]. External pager mechanism allows the use of one or more external pagers in order to provide application specific paging policy. With external pager mechanism, operating system can support QoS for multimedia task and provide flexibility and extensibility. But external pager is only interested in application specific paging policy and memory utilization.

## 3. Transient overload state in virtual memory system

Although the current general-purpose computer system is equipped with much more memory capacity then before, memory requirement for application is not sufficient. As there is much less physical memory than virtual memory, the operating system must be careful that it does not use the physical memory inefficiently. One way to save physical memory is to only load virtual pages that are currently being used by the executing program. This technique of only loading virtual pages into memory as they are accessed is known as demand paging mechanism. Traditionally, general-purpose operating system uses demand paging to load executable images into a processes virtual memory. Whenever a command is executed, the file containing it is opened and its contents are mapped into the processes virtual memory. This is done by modifying the data structures describing this processes memory map and is known as memory mapping. However, only the first part of the image is actually brought into physical memory. The rest of the image is left on disk. As the image executes, it generates page faults and virtual memory system uses the processes memory map in order to determine which parts of the image to bring into memory for execution.

When a process attempts to access a virtual address that is not currently in memory, a page fault is generated. The instruction must be halted, and determine the cause of a page fault. The page fault handler gets control as a result of the machine raising a page fault exception. If the faulting virtual address is invalid this means that the process has attempted to access an invalid virtual address. In this case, currently running process is signaled by kernel, and terminated abnormally. If the faulting virtual address was valid but the page that it refers to is not currently in memory, the operating system must bring the appropriate page into memory from the image on disk. In this case, if there is no unused frame in memory, scan the physical memory for selecting a victim page. If necessary, allocate space on the backing store to receive the contents of the victim page and initiate I/O to write the contents of the victim page to the backing store. After this, adjust the page table for the process to which the victim page belongs to reflect the fact that it is no longer resident in memory. The memory management module locates the page for which the fault was generated on the backing store and initiate I/O to load the page into the page frame. The fetched page is written into a free physical page frame and an entry for the virtual page frame number is added to the processes page table. The process is then restarted at the machine instruction where the memory fault occurred. In page fault handling, a disk access takes a long time, and so the process must wait quite a while until the page has been fetched. If there are other processes that could run then the operating system will select one of them to run. Because page fault handling is urgent kernel task and requires excessive CPU load and I/O load, operating system falls into

transient overloading state.

We show transient overload state in general-purpose operating system while executing diverse real applications such as Netscape, gimp, gzip, find and mpeg_play. Table 1 shows application characteristics and figure 1 shows the required RSS(Resident set size) of each application. The RSS means resident memory frame of each application.

Table 1. Application characteristics (byte)

| | gimp | netscape | gzip | find | mpeg_play |
|---|---|---|---|---|---|
| text | 1683884 | 11616067 | 46829 | 72148 | 138363 |
| data | 138100 | 2013332 | 3072 | 592 | 8120 |
| bss | 347044 | 317412 | 329264 | 604 | 941636 |
| total | 2169028 | 13946811 | 379165 | 73344 | 1088119 |

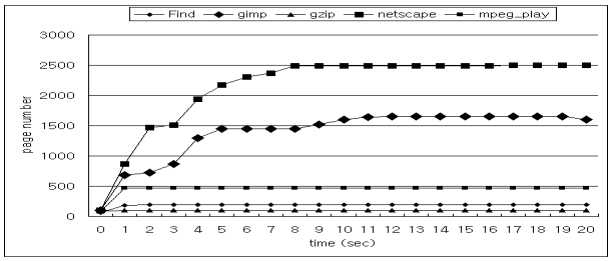


Figure 1. RSS of applications

Figure 2 shows the frequency of page fault when netscape web browser is launched. The x-axis is the elapsed time, and the y-axis is the number of page fault. In this figure, we can find that excessive page fault is occurred when task is launched. Figure 3 shows the RSS of netscape web browser. The result shows that the RSS is increased proportional to page fault frequency. Figure 4 shows the CPU load when netscape is launched. The x-axis is the elapsed time, and the y-axis is the percentage of CPU load. The result shows that excessive page fault lead the system to unstable state that causes temporal overload state.

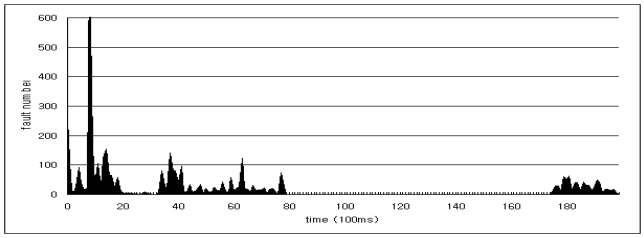


Figure 2. Frequency of page fault in Netscape web browser

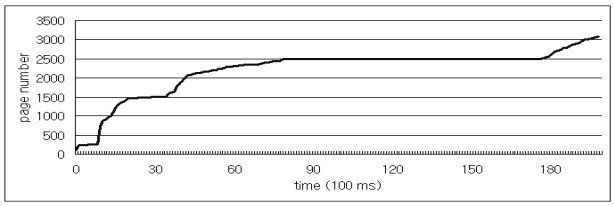


Figure 3. RSS of netscape web browser

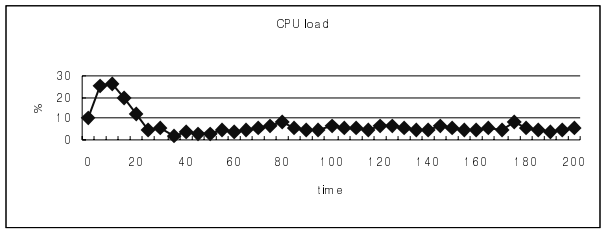


Figure 4. CPU load of Netscape web browser

When transient overload state occurs, real-time task such as multimedia task may miss deadline. Figure 5 represents the deadline miss in the decoding time. We measured the decoding time of a MPEG-1 movie clip with mpeg_play(mpeg_play is a Berkeley MPEG-1 decoder). The x-axis is the video sequence, and the y-axis is the deadline. If y-axis value is under 0, deadline miss occurred in mpeg_play. When mpeg_play is running, we launched netscape web browser about 6 second after. In this figure, the result shows that excessive deadline miss occurred when there is newly launched task.

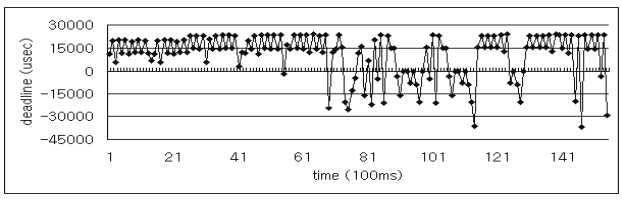


Figure 5. Deadline misses in mpeg_play

## 4. System design

In section 3, It is obvious that an application that fault repeatedly will still degrade the overall system performance. In particular, they will adversely affect the operation of other application. What is required is a system whereby applications benefit from the ability to reserve their execution and not affected by excessive page fault load. In this section, we describe about RAPHA mechanism. A significant feature of the RAPHA algorithm is page fault dispersion that keeps page fault ratio from exceeding available bound by monitoring current system resources. Figure 6 shows RAPHA layout. RAPHA is composed of two stages.:

(1) resource allocation stage : admission controller and resource manager
(2) page fault handling stage

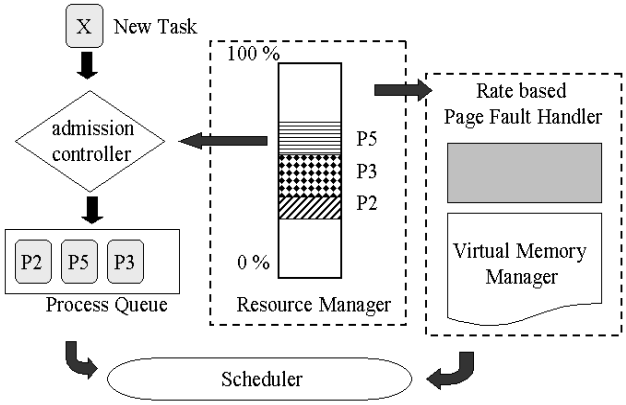


Figure 6. Rate based page fault handling layout

### 4.1 Resource allocation stage

Admission controller abstracts system resource and uses reserves as predictable resource management. In this paper, we used resource reservation mechanism in real-time Mach [1]. Reserves used in real-time Mach prevent applications from over-running their allowed resource usage and interfering with other reserved activities or starving unreserved activities. In real-time Mach, applications reserve capacity on the resources they need to carry out their computations. For example, applications can reserve 10% processor capacity (10ms of computation time on a processor for every 100ms). The application then binds to the reserve, and the processor scheduler uses the information associated with the reserve to control the scheduling of the application. In Real-time Mach, resource reservation algorithm reserves system resource only for application itself and does not accurately count system resource. Because the page fault-handling load is not counted for each application reserve, unexpected virtual memory behavior makes system into unpredictable. In this paper, we try to reserve the kernel activity such as page fault handling. The key role of RAPHA is monitoring system resource and reserves required resource for page fault handling. The proposed algorithm will use the following notation:

-*fault_unit* : processor capacity for processing one page fault
-$task_i$.*pc* : processor capacity for task i.
-$task_i$.*fault_period* : fault processing interval for task i
-$task_i$.*fault_count* : the number of page fault during fault processing interval for task i;
-$task_i$.*fault_limit* : upper limit fault count for task i;
-*system.pc* : sum of all task's processor capacity
-*system.fault_count* : the number of page fault during fault processing interval for system;
-*system.fault_period*: fault processing interval for system
-*system.fault_limit* : sum of all task's fault_limit
-*system.fault_pc* : processor capacity for system page fault

Suppose that *fault_unit* is processor capacity for handling one page fault and *system.fault_pc* is processor capacity for handling page fault, so following Eq. 1 means processor capacity reserve for page fault handling.

$$system.fault\_pc = \frac{system.fault\_limit}{system.fault\_period} * fault\_unit \quad (1)$$

In RAPHA, the sum of processor capacity is less than 1 and satisfy Eq. 2.

$$\sum_{i=0} task_i.pc + system.fault\_pc \le 1 \quad (2)$$

Moreover, from Eq. 1 and Eq. 2, we an conclude that

$$system.fault\_pc \le 1 - \sum_{i=0} task_i.pc \quad (3)$$

From Eq. 3, we have the following corollary.

$$system.fault\_limit = \frac{1 - \sum_{i=0} task_i.pc}{fault\_unit} * system.fault\_period \quad (4)$$

If we assume that *fault_unit* and *system.fault_period* is constant, *system.fault_limit* is a variable that dynamically varied according to the sum of processor capacity of task i. In RAPHA mechanism, we adjust the *system.fault_limit* dynamically.

### 4.2 Page fault handling stage

In RAPHA, when page fault occurred, the number of page fault is counted. If the page fault-handling interval is expired, new fault period is inserted. If excessive page fault is occurred and page fault upper bound is reached, page fault handling is postponed. Figure 7 shows pseudo code of RAPHA.

```
FUNCTION : PAGE_FAULT_HANDLER
  if system.fault_period is expired
      system.fault_period= SYS_FAULT_PERIOD ;
      system.fault_count = 0 ;
  if system.fault_period is not expired
      system.fault_count++ ;
      if system.fault_count is greater than system.fault_limit
          call swapper;
  if task_i.fault_period is expired
      task_i.fault_period= TASK_FAULT_PERIOD ;
      task_i.fault_count = 0 ;
  if task_i.fault_period is not expired
      task_i.fault_count++ ;
      if task_i.fault_count is greater than task_i.fault_limit
          suspend task_i

  end PAGE_FAULT_HANDLER
```

Figure 7. Rate based page fault handler

## 5. Evaluation

In this section, we experimentally evaluate the performance of RAPHA in modified Linux kernel, so called, LMX(Linux Multimedia Extension) that composed of admission control, resource manager and RAPHA mechanism. In particular, we examine the efficacy of the memory resource management mechanisms within RAPHA to (i) keep page fault ratio from exceeding available bound by monitoring current memory resources, (ii) allocate memory bandwidth in a predictable manner, and (iii) provide memory isolation. We use several real applications, benchmarks for our experimental evaluation. We first describe the test-bed for our experiments and then present the results of our experimental evaluation. The test-bed for our experiments consists of 500MHz Pentium II CPU and 64MB RAM and equipped with a 100Mb/s 3-Com Ethernet card. The Linux kernel version in our experiments is based on the 2.2.4. The workload for our experiments consists of combination of real-world applications, benchmarks, and sample applications that we wrote to demonstrate specific features.

A number of simple experiments have been carried out to illustrate the operation of the system so far described. The purpose of these experiments is to show the behavior under same load and how efficiently multimedia task will be executed with RAPHA. We use mpeg_play for multimedia task and measured the performance. Our experimental environment stressed the system with constant CPU load, and measured the page fault frequency. Figure 8 shows excessive page fault occurred without RAPHA while figure 9 shows dispersed page fault frequency. In this experiment, we can conclude that RAPHA efficiently control page fault and get rid of transient overloading state.

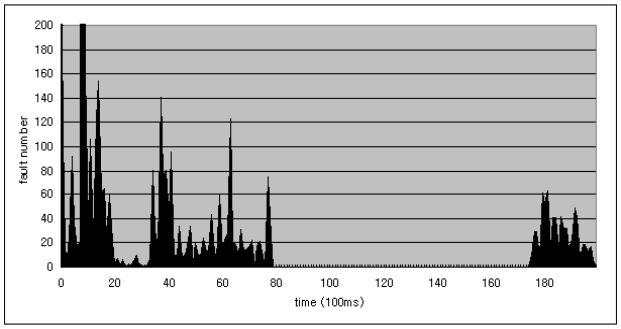


Figure 8. Frequency of netscape page fault(Linux kernel)

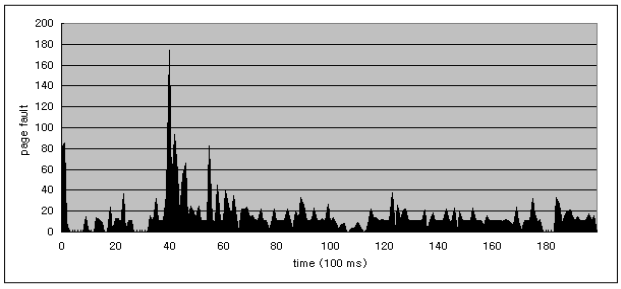


Figure 9. Frequency of netscape page fault(LMX)

Figure 10 shows the resident set of physical page with RAPHA and without RAPHA. The growth of RSS with RAPHA is slowly increased, but the other shows sharp curve. The sharp curve means excessive memory requirements, so if there is no enough memory, paging overhead will be high. Figure 11 shows how multimedia application works. Experiment result shows mpeg_play works well with RAPHA In figure 11, deadline miss rate reduced to 10 % and average delay reduced to 50%. Average delay and miss rate is quite important for multimedia system, because it represents QoS of multimedia application.

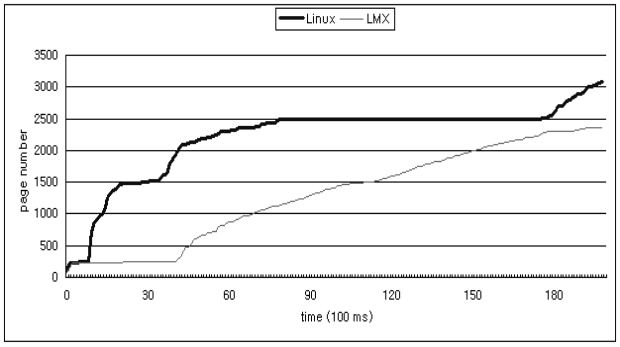


Figure 10. RSS of Netscapeweb browser

| | frame number | deadline miss | miss rate(%) | average delay |
|---|---|---|---|---|
| LINUX | 500 | 161 | 32.2 | 15054 |
| LMX | 500 | 110 | 22 | 6954 |

Figure 11. Deadline miss ratio analysis

## 6. Conclusion

Emerging multimedia and real applications require conventional operating systems to be enhanced along several dimensions. In this paper, we presented the problems of virtual memory when a task is dynamically

launched and how multimedia task influenced. We have argued that the cost of page fault is too high to be hidden from the application, and excessive page fault makes real-time task from isolation. Without isolation, real-time task miss its deadline.

RAPHA employs two key components; the rate based page fault handling and resource reservation for kernel activity. Our experimental results showed that multimedia application could indeed benefit from predictable RAPHA mechanism and application isolation offered. As part of future work, we plan to enhance soft real-time memory management along several dimensions. In particular, we are designing seamless cooperation mechanism among processor, memory and disk.

## References


[1] C. W. Mercer, S. Savage, H. Tokuda, "Processor Capacity reserves: Operating System Support for Multimedia Applications", *Proceedings of the IEEE international Conference on Multimedia Computing and Systems*, Boston, MA, pp. 90-99, May 1994.

[2] Raj Yavatkar, K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications", *In 6th International NOSSDAV Workshop*.

[3] P. Goyal, X. Guo, H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems", *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, WA, PP. 107-122, Oct, 1996.

[4] J, Nieh and Monica S. Lam, "The Design, implementation and Evaluation of SMART : A Scheduler for Multimedia Applications", *Proceedings of 16th ACM Symposium on Operating Systems Principles*, St Malo, France, October, 1997

[5] K. Jeffay, F. D. Smith, A. Moorthy, and J. Anderson, "*Proportional Share Scheduling of Operating System Services for Real-Time Application*", Proc. of Real-Time Systems Symposium, pp. 480-491, Dec. 1998.

[6] I. Stoica, H. Abdel-Wahab, K. Jeffey, S. Baruah, J. Gehrke, and G. Plaxton, "A Proportional Share Resource Allocation Algorithm for Real-Time, Time-shared Systems", *Proc. of Real-Time Systems Symposium*, pp. 288-299, Dec. 1996.

[7] M. B. Jones, D. Rosu, and M-C. Rosu, "CPU Reservation and Time Constraints: Efficient, Predictable Scheduling of Independent Activities", *Proc. of the 16th ACM Symposium on Operating System Principles*, pp. 198-211, Oct. 1997.

[8] Ray Ford, "Concurrent Algorithms for Real-Time Memory Management", *IEEE Software*, Vol.5, Issue 5, pp.10-23, 1988.

[9] G Mapp. An Object-Oriented Approach to Virtual Memory Management. PhD thesis, University of Cambridge Computer Laboratory, January 1992

[10] Steven M. Hand, "Self-Paging in the Nemesis Operating System", *Operating Systems Design and Implementation*, 73-86, 1999.

[11] Kieran Harty and David R. Cheriton, "Application-controlled physical memory using external page-cache management", In *Proc. Fifth International Conf. On Architectural Support for Programming Languages and Operating systems, SIGOPS Operating Systems Review Special Issue*, volume 26, page 187, Boston, MA, October 12-15 1992.

[12] CHao-Hsien Lee, Meng Chang Chen, and RueiChuan Chang. "HiPEC: High performance external virtual memory caching", In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 153-164, November 1994.

[13] Dylan McNamee and Katherine Armstrong. "Extending The Mach External Pager Interface To Accommodate User-Level Page Replacement Policies", *In Proceedings of the USENIX Association Mach Workshop*, pages 17-29, 1990.

[14] M. Young, A. Tevanian, R. Rashid, D. Golub, J. Eppinger, J. Chew, W. Bolosky, D. Black, and R. Baron. "The duality of memory and communication in the implementation of a multiprocessor operating system", *Proceedings of the 11th Symposium on Operating System Principles*, pages 63--76, December 1987.

[15] Dawson R. Engler and M. Frans Kaashoek and James O'Toole, Exokernel: An Operating System Architecture for Application-Level Resource Management, SOSP 251-266,1995

[16] S. F. Kaplan, "Compressed Caching and Modern Virtual Memory Simulation", *Ph.D. Thesis, University of Texas at Austin*, December 1999.

[17] B. Bershad et al. Extensibility, safety and performance in the SPIN operating system. *In Proc. 15-th Symposium on Operating Systems Principles*, pages 267--284, December 1995.

[18] Small, C., Seltzer, M., "VINO: An Integrated Platform for Operating System and Database Research," *Harvard University Computer Science Technical Report TR-30-94* ,1994.

[19] Richard P. Draves, Gilad Odinak, Scott M. Cutshall. "The Rialto Virtual Memory System". Technical Report MSR-TR-97-04, *Microsoft Research*, February, 1997.

[20] J. Bruno, E. Gabber, B. zden, and A. Silberschatz, "The Eclipse Operating System: Providing Quality of Service via Reservation Domains", *Proceedings of the USENIX*

[21] Jim Mauro and Richard Mc Dougall, "*SOLARIS Internals"*, SUN Microsystems, Inc, 2001.

[22] Michael Beck, et al. "*Linux Kernel Internals second Edition"*, Addison-Wesley, 1998.