Preface

Operating Systems research is a vital and dynamic field. Even young computer science students know that Operating Systems are the core of any computer system and a course about Operating Systems is more than common in any Computer Science department all over the world.

This book aims at introducing subjects in the contemporary research of Operating Systems. One-processor machines are still the majority of the computing power far and wide. Therefore, this book will focus at these research topics i.e. Non-Distributed Operating Systems. We believe this book can be especially beneficial for Operating Systems researchers alongside encouraging more graduate students to research this field and to contribute their aptitude.

A probe of recent operating systems conferences and journals focusing on the "pure" Operating Systems subjects (i.e. Kernel's task) has produced several main categories of study in Non-Distributed Operating Systems:

- Kernel Security and Reliability
- Efficient Memory Utilization
- Kernel Security and Reliability
- I/O prefetching
- Page Replacement Algorithms

We introduce subjects in each category and elaborate on them within the chapters. The technical depth of this book is definitely not superficial, because our potential readers are Operating Systems researchers or graduate students who conduct research at Operating System labs. The following paragraphs will introduce the content and the main points of the chapters in each of the categories listed above.

KERNEL SECURITY AND RELIABILITY

Kernel Stack Overflows Elimination

The kernel stack has a fixed size. When too much data is pushed upon the stack, an overflow will be generated. This overflow can be illegitimately utilized by unauthorized users to hack the operating system. The authors of this chapter suggest a technique to prevent the kernel stack from overflowing by using a kernel stack with a flexible size.

Device Driver Reliability

Device Drivers are certainly the Achilles' heel of the operating system kernel. The writers of the device drivers are not always aware of how the kernel was written. In addition, many times, only few users may have a given device, so the device driver is actually not indeed battle-tested. The author of this chapter suggests inserting an additional layer to the kernel that will keep the kernel away from the device driver failures. This isolation will protect the kernel from unwanted malfunctions along with helping the device driver to recover.

Identifying Systemic Threats to Kernel Data: Attacks and Defense Techniques

Installing a malware into the operating system kernel by a hacker can has devastating results for the proper operation of a computer system. The authors of this chapter show examples of dangerous malicious code that can be installed into the kernel. In addition, they suggest techniques how to protect the kernel from such attacks.

EFFICIENT MEMORY MANAGEMENT

Swap Token: Rethink the Application of the LRU Principle on Paging to Remove System Thrashing

The commonly adopted approach to handle paging in the memory system is using the LRU replacement algorithm or its approximations, such the CLOCK policy used in the Linux kernels. However, when a high memory pressure appears, LRU is incapable of satisfactorily managing the memory stress and a thrashing can take place. The author of this chapter proposes a design to alleviate the harmful effect of thrashing by removing a critical loophole in the application of the LRU principle on the memory management.

Application of both Temporal and Spatial Localities in the Management of Kernel Buffer Cache

With the objective of reducing the number of disk accesses, operating systems usually use a memory buffer to cache previously accessed data. The commonly used methods to determine which data should be cached are utilizing only the temporal locality while ignoring the spatial locality. The author of this chapter proposes to exploit both of these localities in order to achieve a substantially improved I/O performance, instead of only minimizing number of disk accesses.

Alleviating the Trashing by Adding Medium-Term Scheduler

When too much memory space is needed, the CPU spends a large portion of its time swapping pages in and out the memory. This effect is called Thrashing. Thrashing's result is a severe overhead time and as a result a significant slowdown of the system. Linux 2.6 has a breakthrough technique that was suggested

by one of these book editors - Dr. Jiang and handles this problem. The authors of this chapter took this known technique and significantly improved it. The new technique is suitable for much more cases and also has better results in the already handled cases.

KERNEL FLEXIBILITY

The Exokernel Operating System and Active Networks

The micro-kernel concept is very old dated to the beginning of the seventies. The idea of micro-kernels is minimizing the kernel. I.e. trying to implement outside the kernel whatever possible. This can make the kernel code more flexible and in addition, fault isolation will be achieved. The possible drawback of this technique is the time of the context switches to the new kernel-aid processes. Exokernel is a micro-kernel that achieves both flexibility and fault isolation while trying not to harm the execution time. The author of this chapter describes the principles of this micro-kernel.

I/O PREFETCHING

Exploiting Disk Layout and Block Access History for I/O Prefetch

Prfetching is a known technique that can reduce the fetching overhead time of data from the disk to the internal memory. The known fetching techniques ignore the internal structure of the disk. Most of the disks are maintained by the Operating System in an indexed allocation manner meaning the allocations are not contiguous; hence, the oversight of the internal disk structure might cause an inefficient prefetching. The authors of this chapter suggests an improvement to the prefetching scheme by taking into account the data layout on the hard disk.

Sequential File Prefetching in Linux

The Linux operating system supports autonomous sequential file prefetching, aka readahead. The variety of applications that Linux has to support requires more flexible criteria for identifying prefetchable access patterns in the Linux prefetching algorithm. Interleaved and cooperative streams are example patterns that a prefetching algorithm should be able to recognize and exploit. The author of this chapter proposes a new prefetching algorithm that is able to handle more complicated access patterns. The algorithm will continue to optimize to keep up with the technology trend of escalating disk seek cost and increasingly popular multi-core processors and parallel machines.

PAGE REPLACEMENT ALGORITHMS

Adaptive Replacement Algorithm Templates and EELRU

With the aim of facilitating paging mechanism, the operating system should decide on "page swapping out" policy. Many algorithms have been suggested over the years; however each algorithm has advantages and disadvantages. The authors of this chapter propose to adaptively change the algorithm according to

the system behavior. In this way the operating system can avoid choosing inappropriate method and the best algorithm for each scenario will be selected.

Enhancing the Efficiency of Memory Management in a Super-Paging Environment by AMSQM

The traditional page replacement algorithms presuppose that the page size is a constant; however this presumption is not always correct. Many contemporary processors have several page sizes. Larger pages that are pointed to by the TLB are called Super-Pages and there are several super-page sizes. This feature makes the page replacement algorithm much more complicated. The authors of this chapter suggest a novel algorithm that is based on recent constant page replacement algorithms and is able to maintain pages in several sizes.

This book contains surveys and new results in the area of Operating System kernel research. The books aims at providing results that will be suitable to as many operating systems as possible. There are some chapters that deal with a specific Operating System; however the concepts should be valid for other operating systems as well.

We believe this book will be a nice contribution to the community of operating system kernel developers. Most of the existing literature does not focus on operating systems kernel and many operating system books contain chapters on close issues like distributed systems etc. We believe that a more concentrated book will be much more effective; hence we made the effort to collect the chapters and publish the book.

The chapters of this book have been written by different authors; but we have taken some steps like clustering similar subjects to a division, so as to make this book readable as an entity. However, the chapters can also be read individually. We hope you will enjoy the book as it was our intention to select and combine relevant material and make it easy to access.