CS443 Paper Review
Lei Yang
2005-4-25

****

Title

Virtual Memory Primitives for User Programs

Author

Andrew W. Appel and Kai Li

Summary

This paper surveyed several user-level algorithms that make use of virtual memory page-protection techniques and analyzed their common characteristics, in order to identify what virtual memory primitives the operating system should provide to user processes.

Most important ideas

The motivation of this survey paper is based on such an observation: virtual memory no longer serves for only traditional purpose, i.e., extending the address space and protecting user processes from one another; instead, it has evolved into a user level component of a hardware and operating system interface. Intuitively, one might ask: if this is the case, what primitives should VM provide to user processes to improve the efficiency of such applications? To answer this question, the paper looked into several algorithms (or applications, I'd say) that rely on virtual memory primitives, which have not been paid enough attention in the past. The investigated algorithms are: concurrent garbage collection, shared virtual memory, concurrent checkpointing, general garbage collection, persistent stores, extending addressability, data-compression paging, and heap overflow detection. Almost all the above algorithms fall into one of the two categories: protecting pages in large batches then upon each page fault trap unprotecting one page, and protecting a page and unprotecting a page individually. The authors performed measurements on Ultrix, SunOS, and Mach with several different platforms, in the execution of two specially designed benchmark programs: accessing a random protected page and in the fault handler, protecting some other page, and unprotecting the faulting page; protecting 100 pages, accessing each page in a random sequence, and in the fault handler, unprotecting the faulting page. The results lead to this conclusion: operating system designers should pay special attention to the page protection and fault handling efficiency. Page size is another important parameter. In addition, for many algorithms the configuration of TLB hardware may not be particularly important.

Flaws/Questions

I think this paper provides a guide to operating system designers from a perspective that is different from conventional ones. It pointed out "what should be done" in VM design to provide more efficient interface to user processes that take advantages of VM primitives, but didn't actually tell the readers "How". It did indicate that although all the algorithms described in the paper (except heap overflow detection) share five or more of the common characteristics, big variations exist in the usages and requirements of the algorithms. For example, some protect pages one at a time, while others protect pages in large batches; some use memory protection only to keep track of modified pages, while others require access to protected pages when run concurrently. How to solve the problems introduced by different types of algorithms and make the system adaptively efficient still remains a question.

Relevance/Potential Future Research

There is later work (in 2002) on fine-grained protection, that claims to offer better performance with the algorithms (applications) described in this paper. It's also interesting to find that the main strength of this paper, improving VM page protection and fault handling, has not been fully investigated in following research; however, the sample algorithms(applications) that the authors described which use virtual memory primitives in place of software tests, special hardware , or microcode, have been cited and implemented by quite a few researchers (e.g., swap compression, garbage collection).