Flattening the Page Tables on Linux in x86_64

Rubin Du*
University of Illinois at Urbana-Champaign
rd25@illinois.edu

Shanbo Zhang*
University of Illinois at Urbana-Champaign
shanboz2@illinois.edu

Abstract

The abstract of the project report

1 Introduction

When the computing system was first invented, the memory was limited and expensive, and relevant logic was simple, because there were only one program running on the CPU. As time went on, the memory became larger and cheaper, and multiple users might run their programs on the same computers, but the CPU could only run one program at a time. The operating system should be able to take control of the CPU and manage the memory access of the user programs. Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory and creates the illusion to users of a very large (virtual) main memory. Paging scheme partitions the physical memory into fixed-size blocks called frames, and uses a table to map each virtual page to a physical frame. The page table contains information about each page.

Originally, the page table in x86 architecture is a two-level page table, where the address field in the first level points to the starting address of the second level page table, and the address field in the second level page table points to the starting address of the physical frame. Then, in x86_64 architecture, the page table is extended to four levels. Later, it also supports 5-level page table extension. Typically, the page table is stored in memory due to its large size. For a memory access, in the worst case, the CPU needs to walk through all the levels of the page table to find the physical address. This is inefficient, and it could be the bottleneck of the system performance. Because of the popularity of machine learning, which consists of complex memory access patterns, the page table walking could be a performance bottleneck.

In this project, we implements a new page table structure called flattened page table (FPT). FPT maintains the logical behavior and interface of the traditional multi-level tree-structured page table, while enabling the selective flattening of adjacent levels to reduce the depth of the page table walk. Specifically, assuming the 5-level paging extension is not used, we support three folding modes: L4L3, L3L2, and L4L3-L2L1 folding. In each mode, the virtual address range covered by individual levels remains unchanged, but the number of levels traversed per walk is reduced by merging multiple levels into a single larger page table. The resulting structure remains logically hierarchical, but folded levels are stored as wider tables in memory, decreasing the number of required memory accesses for translation. Take L3L2 folding as an example. An L3 page table page is 4KB and maps a 1GB address space, and an L2 page table page is 4KB and maps a 2MB address space. In FPT L3L2 mode, however, the level 3 no longer exists, and L2 page table page is 2MB and maps a 1GB address space. Each entry in the L2 page table page points to an L1 page table page.

2 Middle Sections

You can have as many middle sections as possible based on your organization of the report.

3 Related Work

The related work of your project [1].

4 Conclusion

This project is awesome.

5 Metadata

The presentation of the project can be found at:

https://zoom/cloud/link/

The code/data of the project can be found at:

https://github.com/you/repo

References

[1] Dijkstra, E. W. The Structure of the "THE" Multiprogramming System. In *Proceedings of the 1st ACM Symposium on Operating System Principles (SOSP'67)* (Oct. 1967).

*Both authors contributed equally.