Page Size Extension in JOS

Jianing Yang and Xiaochuang Han
Georgia Institute of Technology

Intro

This project is intended to give more freedom to the JOS memory management system. Specifically, we enabled JOS to support Intel x86 4MB pages, allowing both 4KB pages and 4MB pages to be allocated and mapped.

Background

Processors after Intel 80386 have added a PTE_PS bit in their page directory to support 4MB “super pages” in addition to regular 4KB pages. However, our JOS did not support super pages. By adding support for super pages in JOS, we gave more freedom to kernel programs of deciding which page structure to use. This will be particularly useful for kernel programs that require a large contiguous chunk of memory.

Problem

Enable x86 Page Size Extension

With the out-of-the-box settings of Intel x86, super pages are not supported by the processor. We have to figure out a way to enable this setting in the processor.

Use super pages in entry page directory

Recall that in kern/entrypgdir.c, JOS used a long (1024 lines of code) manually defined table to map the kernel. This is because JOS did not support super pages and it had to use a second-level page table to map 1024 regular 4KB pages. After we enable page size extension, we should be able to replace these 1024 4KB pages with one single super page.

Divide up physical memory

JOS had all its physical memory divided into 4KB regular pages, Now with pages and super pages co-existing in the system, we need to figure out how to divide the physical memory to dedicate a chunk of memory to be super pages.

Create super page structures and related functions

Then, just like regular pages, we need to build the infrastructures to keep track of super pages. These includes:

size_t nsuper_pages
struct PageInfo *super_pages
struct PageInfo *super_page_free_list
super_page_alloc()
super_page_free()

Support super page in kernel page directory

We also need functions to map and free the allocated super page in the kernel page directory. These includes:

super_page_insert()
super_page_remove()

These functions should handle different scenarios to allow both regular pages and super pages to be mapped in the kernel page directory.

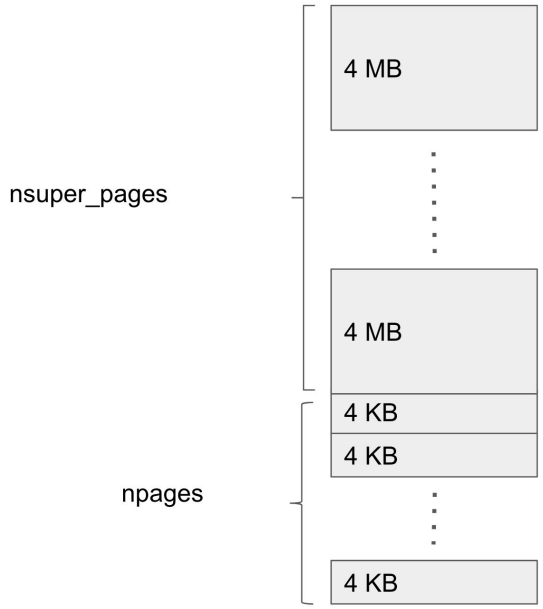
Write test code and test our implementation
We should write test code to test the correctness of our implementation regarding the problems mentioned above.

Approach

Enable x86 Page Size Extension
In kern/entry.S, we set the PSE bit in the CR4 register to tell the processor we are using page size extension.

Use super pages in entry page directory
In kern/entrypgdir.c, we changed the mapping in entry_pgdir to map 2 super pages instead of 2048 regular pages.

Divide up physical memory



Originally, in i386_detect_memory() of kern/pmap.c, JOS divided up its physical memory into all 4KB pages. First, for the convenience of implementation, we rounded down available physical memory to be super-page-aligned. For example, if the original available physical memory is 17 MB, we only treat it as if there is only 16 MB and discard the top 1 MB memory. Then, to add super pages to the system, we cut out the top 4096 4KB pages and dedicated these physical

memories as the memory that super pages would reside in. In other words, npages is now reduced by 4096 (not counting memory reduced by the rounddown) and nsuper_pages is 4.

Create super page structures and related functions
In kern/pmap.c, we added the following variables:

size_t nsuper_pages
struct PageInfo *super_pages
struct PageInfo *super_page_free_list

Similar to the variables used to keep track the status of regular pages, these variables are meant to keep track the status of super pages.

In kern/pmap.c, we also added the following functions:
super_page_alloc()
super_page_free()

These two functions are almost identical to page_alloc() and page_free() except that they operate on super_page_free_list instead of page_free_list.

Support super page in kernel page directory
To support both super page and regular page in the kernel page directory, we first need to align every entry to PTSIZE. As described below, we have 6 cases when inserting a super page or a regular page.

Case 1: Insert a regular page to a blank slot
We first walk down 2 layers and get a page table entry. Then we can directly put the new page's address into the slot.

Case 2:Insert a super page to a blank slot
We first walk down 1 layer and get a page directory entry. Then we can directly put the new super page's address into the slot.

Case 3:Insert a regular page to a slot where a regular page already exists
We walk down 2 layers and get a page table entry, remove the current page table entry, and put the new page's address into the slot.

Case 4:Insert a super page to a slot where a super page already exists
We walk down 1 layer and get a page directory entry, remove the current page directory entry, and put the new super page's address into the slot.

Case 5:Insert a regular page to a slot where a super page already exists
We walk down 1 layer and get a page directory entry, remove the current page directory entry, create a page table at that slot, and put the new page's address into an appropriate slot of the created page table.

Case 6:Insert a super page to a slot where a regular page already exists
We walk down 1 layer and get a page directory entry (page table), iteratively remove all present entries in that page table, and put the new super page's address into the cleaned slot.

With the above 6 cases in mind, we created two entry functions for inserting a page: page_insert() for inserting a regular page and super_page_insert() for inserting a super page. Here's the pseudo pseudo code.

page_insert():
    If we find a regular page directory entry in place:
        pgdir_walk() to get the right page table entry
        If that entry is marked as present:
            page_remove() to remove that entry
        Fill the entry with the new address with correct permissions
    If we find a super page directory entry in place:
        super_pgdir_walk() to get the right page directory entry
        If that entry is marked as present:
            super_page_remove() to remove that entry
        pgdir_walk() to create a page table at the place and get the right page table entry
        Fill the entry with the new address with correct permissions
super_page_insert():
    super_pgdir_walk() to get the right page directory entry
    If that entry is marked as present:
        super_page_remove() to remove anything in that slot (super page or page table)
    Fill the entry with the new address with correct permissions

Write test code and test our implementation
We designed the following test to make sure all the code we wrote can function as expected:

1. page_alloc() and super_page_alloc() a bunch of pages and super pages
2. memset() those pages and super pages with a pre-defined magic number
3. page_insert() and super_page_insert() the allocated pages and super pages to the simulate the 6 cases mentioned above
4. read with the virtual address and check the magic number

Results
Our project mainly achieved four results:

Eliminate entry page tables
We eliminated the long entry page table array by using super page mapping. This little trick helped us save 1024 lines in *entrypgdir.c*.

Allocate and free super pages in kernel memory
We made it possible to allocate and free super pages in the kernel memory. The super page system keeps every feature of the regular page system except that its size is 1000 times larger.

Share the kernel page directory between regular pages and super pages
This is an important achievement. In order that the kernel programs could use the regular pages and the super pages normally, we have to let the regular pages and the super pages share one kernel page directory. As described in the previous section, our page inserting function and page removing function make sure that the system works well under this share.

Compatible with all other components
We did not only make the paging system work well. All other key components of an operating system, such as environment, interrupt, scheduler, and file system, work normally under our project. The modified JOS system can start up and run programs just like the original one.

Conclusions

Our project gives more freedom to the JOS memory management system. We successfully make it possible for the kernel programs to choose from different sized pages. This will be particularly useful for kernel programs that require a large contiguous chunk of memory. For future works, we will try to add system calls so that the user space could also utilize our different page size feature.

Appendix

Code Repository
The code repository is a private repository on Georgia Tech github. Send an email to jyang416@gatech.edu to request access.
Code repository: https://github.gatech.edu/cs3210-yang-han/final-project