CS 130 Project 2: User Programs
Design Document

Tianyi Zhang
2018533074
zhangty2@shanghaitech.edu.cn

Haoran Dang
2018533259
danghr@shanghaitech.edu.cn

Derun Li
2018533152
lidr@shanghaitech.edu.cn

0 PRELIMINARIES

0.1 Preliminary Comments

No preliminary comment for this project.

0.2 References

- Pintos Guide by Stephen Tsung-Han Sher: https://static1.squarespace.com/static/5b18aa0955b02c1de94e4412/t/5e1bb4809e4b0a78012be132/1578874001361/Sher%282016%29_Pintos_Guide

1 PAGE TABLE MANAGEMENT

1.1 Data Structures

1.1.1 Copy here the declaration of each new or changed `struct` or `struct` member, global or static variable, `typedef`, or enumeration. Identify the purpose of each in 25 words or less.

1.2 Algorithms

1.2.1 In a few paragraphs, describe your code for accessing the data stored in the SPT about a given page.

1.2.2 How does your code coordinate accessed and dirty bits between kernel and user virtual addresses that alias a single frame, or alternatively how do you avoid the issue?

1.3 Synchronization

1.3.1 When two user processes both need a new frame at the same time, how are races avoided?

1.4 Rationale

1.4.1 Why did you choose the data structure(s) that you did for representing virtual-to-physical mappings?

2 PAGING TO AND FROM DISK

2.1 Data Structures

2.1.1 Copy here the declaration of each new or changed `struct` or `struct` member, global or static variable, `typedef`, or enumeration. Identify the purpose of each in 25 words or less.

2.2 Algorithms

2.2.1 When a frame is required but none is free, some frame must be evicted. Describe your code for choosing a frame to evict.

2.2.2 When a process P obtains a frame that was previously used by a process Q, how do you adjust the page table (and any other data structures) to reflect the frame Q no longer has?

2.2.3 Explain your heuristic for deciding whether a page fault for an invalid virtual address should cause the stack to be extended into the page that faulted.

2.3 Synchronization

2.3.1 Explain the basics of your VM synchronization design. In particular, explain how it prevents deadlock. (Refer to the textbook for an explanation of the necessary conditions for deadlock.)

2.3.2 A page fault in process P can cause another process Q's frame to be evicted. How do you ensure that Q cannot access or modify the page during the eviction process? How do you avoid a race between P evicting Q's frame and Q faulting the page back in?

2.3.3 Suppose a page fault in process P causes a page to be read from the file system or swap. How do you ensure that a second process Q cannot interfere by e.g. attempting to evict the frame while it is still being read in?

2.3.4 Explain how you handle access to paged-out pages that occur during system calls. Do you use page faults to bring in pages (as in user programs), or do you have a mechanism for 'locking' frames into physical memory, or do you use some other design? How do you gracefully handle attempted accesses to invalid virtual addresses?

2.3.5 Rationale.

2.3.6 A single lock for the whole VM system would make synchronization easy, but limit parallelism. On the other hand, using many locks complicates synchronization and raises the possibility for deadlock but allows for high parallelism. Explain where your design falls along this continuum and why you chose to design it this way.

3 MEMORY MAPPED FILES

3.1 Data Structures

3.1.1 Copy here the declaration of each new or changed `struct` or `struct` member, global or static variable, `typedef`, or enumeration. Identify the purpose of each in 25 words or less.

3.2 Algorithms

3.2.1 Describe how memory mapped files integrate into your virtual memory subsystem. Explain how the page fault and eviction processes differ between swap pages and other pages.

3.2.2 Explain how you determine whether a new file mapping overlaps any existing segment.

3.3 Rationale

3.3.1 Mappings created with 'mmap' have similar semantics to those of data demand-paged from executables, except that 'mmap' mappings are written back to their original files, not to swap. This implies that much of their implementation can be shared. Explain why your implementation either does or does not share much of the code for the two situations.

4 SURVEY QUESTIONS

In your opinion, was this assignment, or any one of the two problems in it, too easy or too hard? Did it take too long or too little time?

Did you find that working on a particular part of the assignment gave you greater insight into some aspect of OS design?

Is there some particular fact or hint we should give students in future quarters to help them solve the problems? Conversely, did you find any of our guidance to be misleading?

Do you have any suggestions for the TAs to more effectively assist students, either for future quarters or the remaining projects?

Any other comments?

CONTRIBUTORS

Task		Tianyi Zhang	Haoran Dang	Derun Li
Task 1 Page Table Management	Concept	✓	✓	✓
	Implementation	✓	✓	✓
	Debugging	✓	✓	✓
	Design Document	✓	✓	✓
Task 2 Paging to and From Disk	Concept	✓	✓	✓
	Implementation	✓	✓	✓
	Debugging	✓	✓	✓
	Design Document	✓	✓	✓
Task 3 Memory Mapped Files	Concept	✓	✓	✓
	Implementation	✓	✓	✓
	Debugging	✓	✓	✓
	Design Document	✓	✓	✓