세계 속의 GIST 이공계 대학

Operating System Project 4

Release Date: 2024-05-13

Due Date: 2024-05-27

0. Grading Criteria

- This assignment will be graded through the ‘make check’ command, just like Task 2. But only one difference is now I run ‘make check’ in ‘**pintos/src/vm**’.
- Before this task, you must fix any bugs in your project 3. Because those bugs will most likely cause the same problems in this project.

  I will upload project 3 solution on LMS, so download it if you need the solution.
- In this assignment, you will modify existing code to support demand paging (i.e. handling of page faults will have to change).

- Tips :

- Just like the last assignment, don't try to pass all the tests. There is no big minus point if you get a few tests fail. Rather than giving up, aim to pass as many tests as possible.

0. Grading Criteria - Rules

- All problems (4 problems)
  - Report: 1pt
  - Test result: # of passed tests / # of all tests(33 vm tests) * 4pts
  - If you have any failed test on 76 Project 3 tests, it will be considered as failed of the vm test.
  - Test result won’t go below zero.

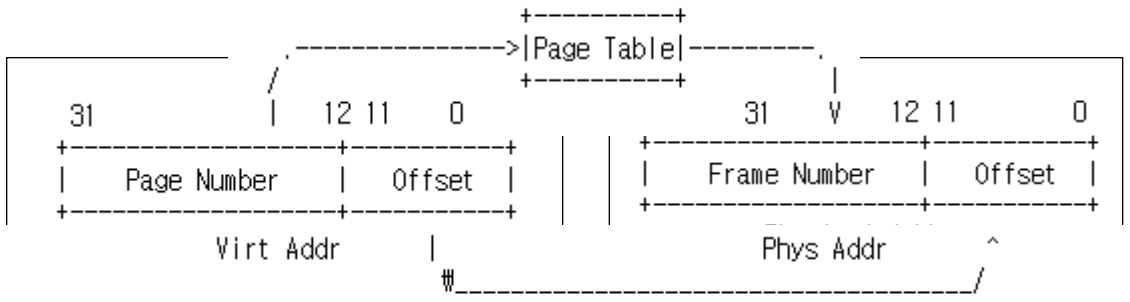
- If you have any questions, please mail me at minwook-lee@gm.gist.ac.kr

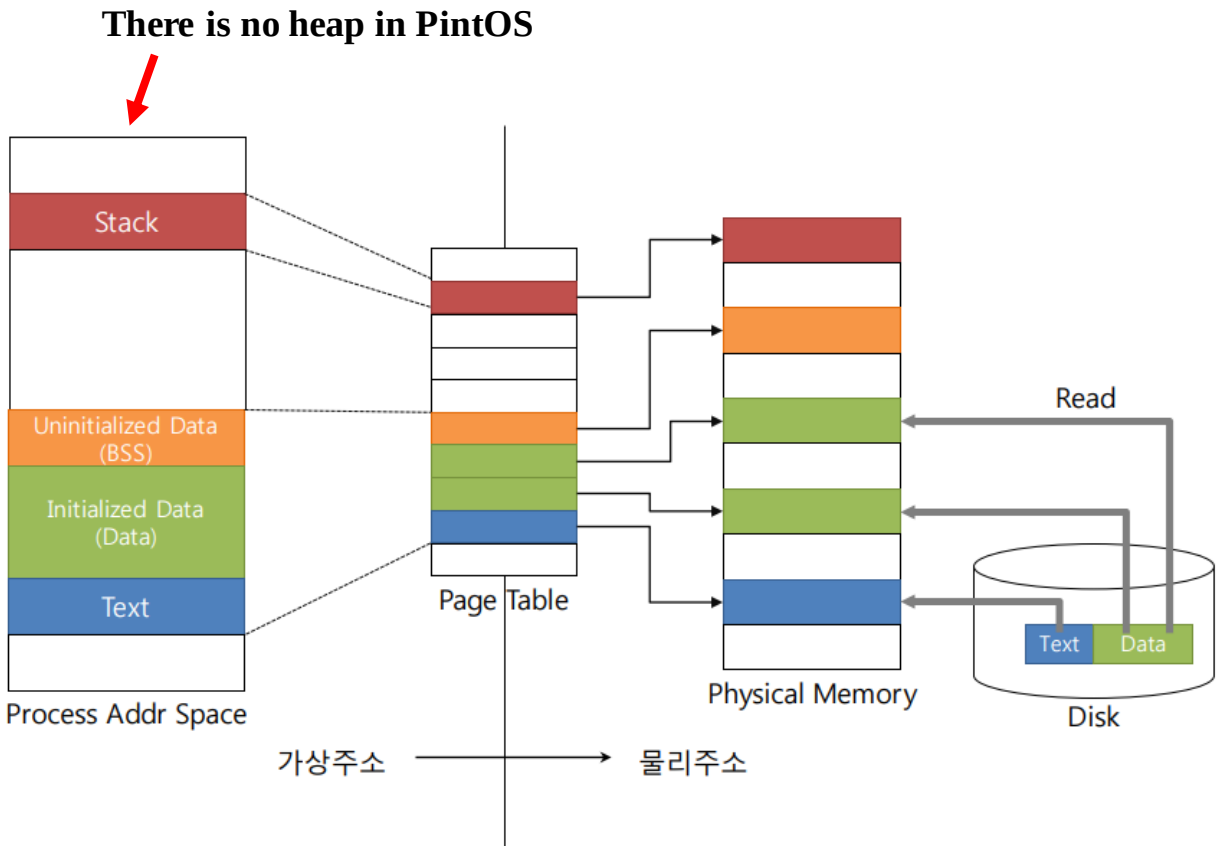
1. Project #4 Introduction

- In this assignment, you need to enable programs to deal with **Virtual Memories**.
- You will mainly work in the "*~/pintos/src/vm*" directory, but you will also be interacting with almost every other part of pintos
- For this assignment, you need to know about Pages, Frames, Demand paging, and Virtual memories.
- While doing this project, your kernel should pass all the project 3 test cases after any steps.
- Use hex_dump() function to check data.
- Do not forget:
  - All the instructions are in the pintos Stanford manual. **Please read the manual**.

Problem 1.

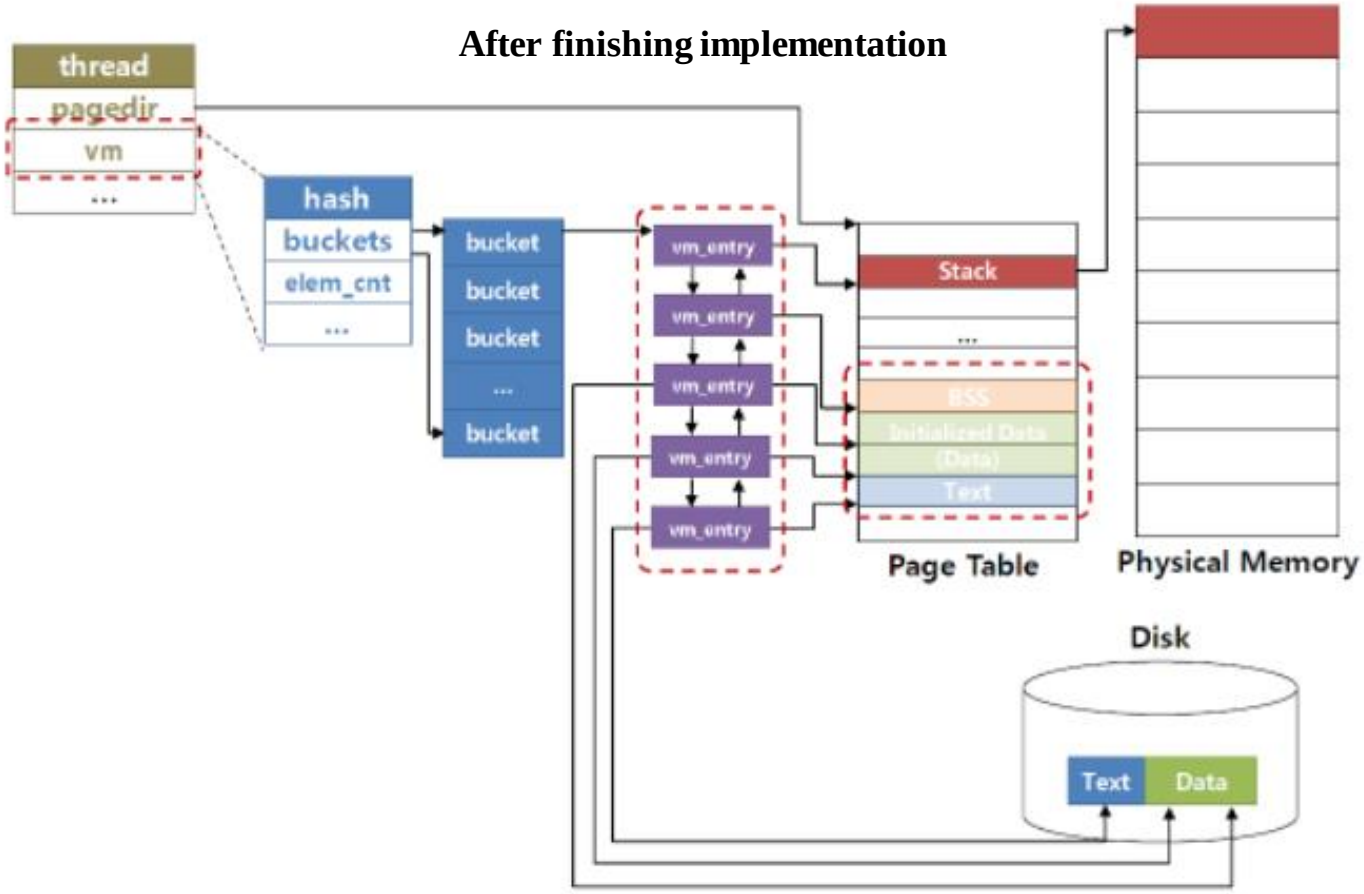
- **Paging (required, 1pt)**
- Problem Manual: Current state
  - There is no virtual memory in PintOS system.
  - Physical memory directly connected to address.
  - Swapping action is not available now.





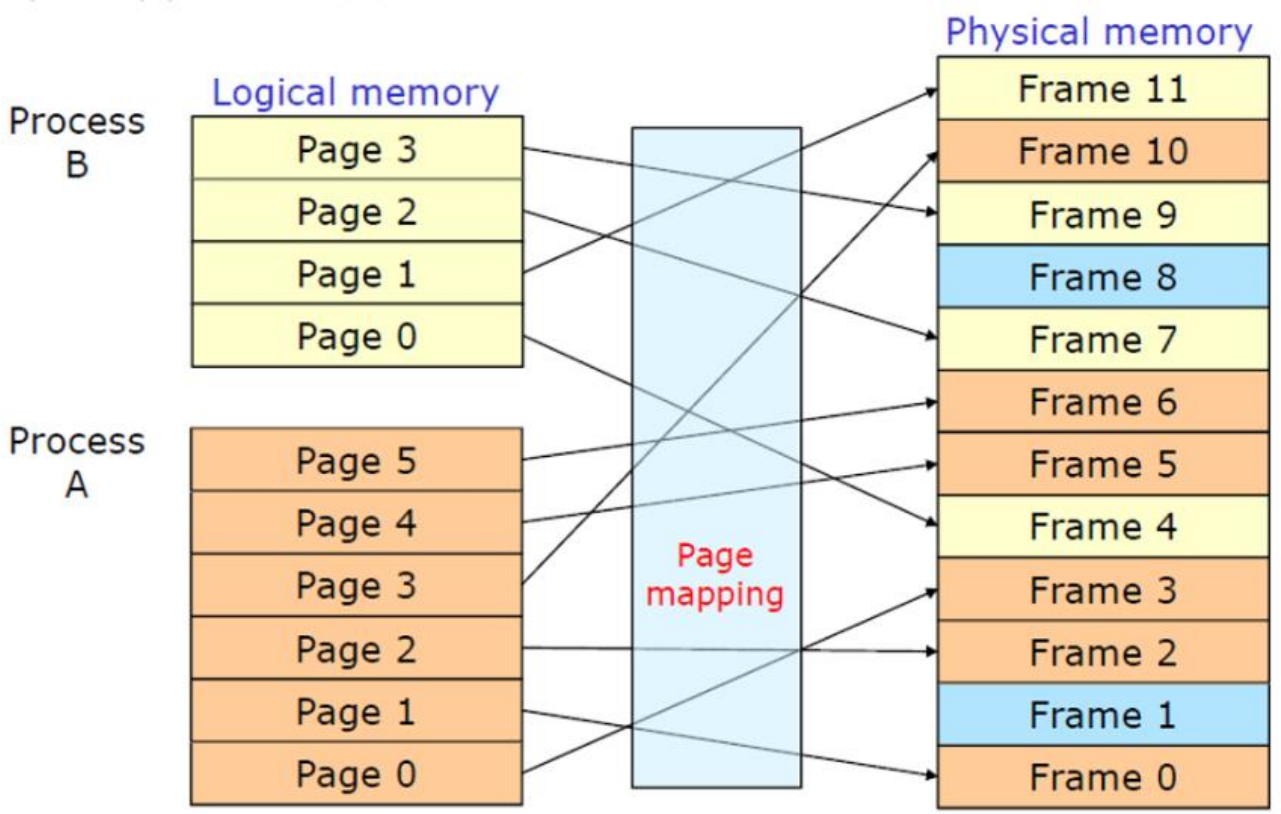
Problem 1.

- **Paging (required, 1pt)**
- Problem Manual:



Problem 1.

- **Paging (required, 1pt)**
- **Requirement:**
  - Implement paging for segments loaded from executables.
  - Implement global page replacement algorithm that approximates LRU algorithm.
  - Your implemented algorithm should perform at least as well as simple variant of “second chance” or “clock” algorithm.

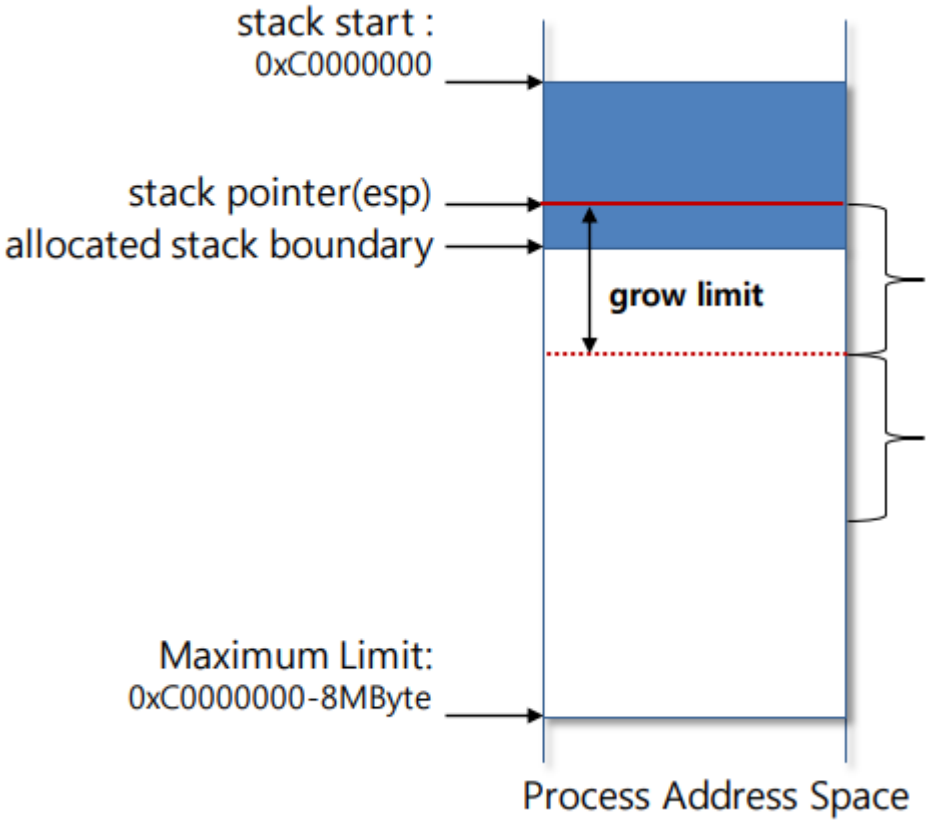


Problem 2.

- **Stack Growth (required, 1pt)**
- **Requirement:**
  - In the previous project, the stack was a single page at the top of the user virtual address space.
  - Now, if the stack grows past its current size, allocate additional pages as necessary.
  - Current stack size is 4KB, you need to expand this at most 8MB

Problem 2.

- **Stack Growth (required, 1pt)**
- Problem Manual
  - If there is a access over the current stack size.
  - Check its valid address access or not first.
  - If its valid, then expand the limit.
  - If not return segmentation fault.

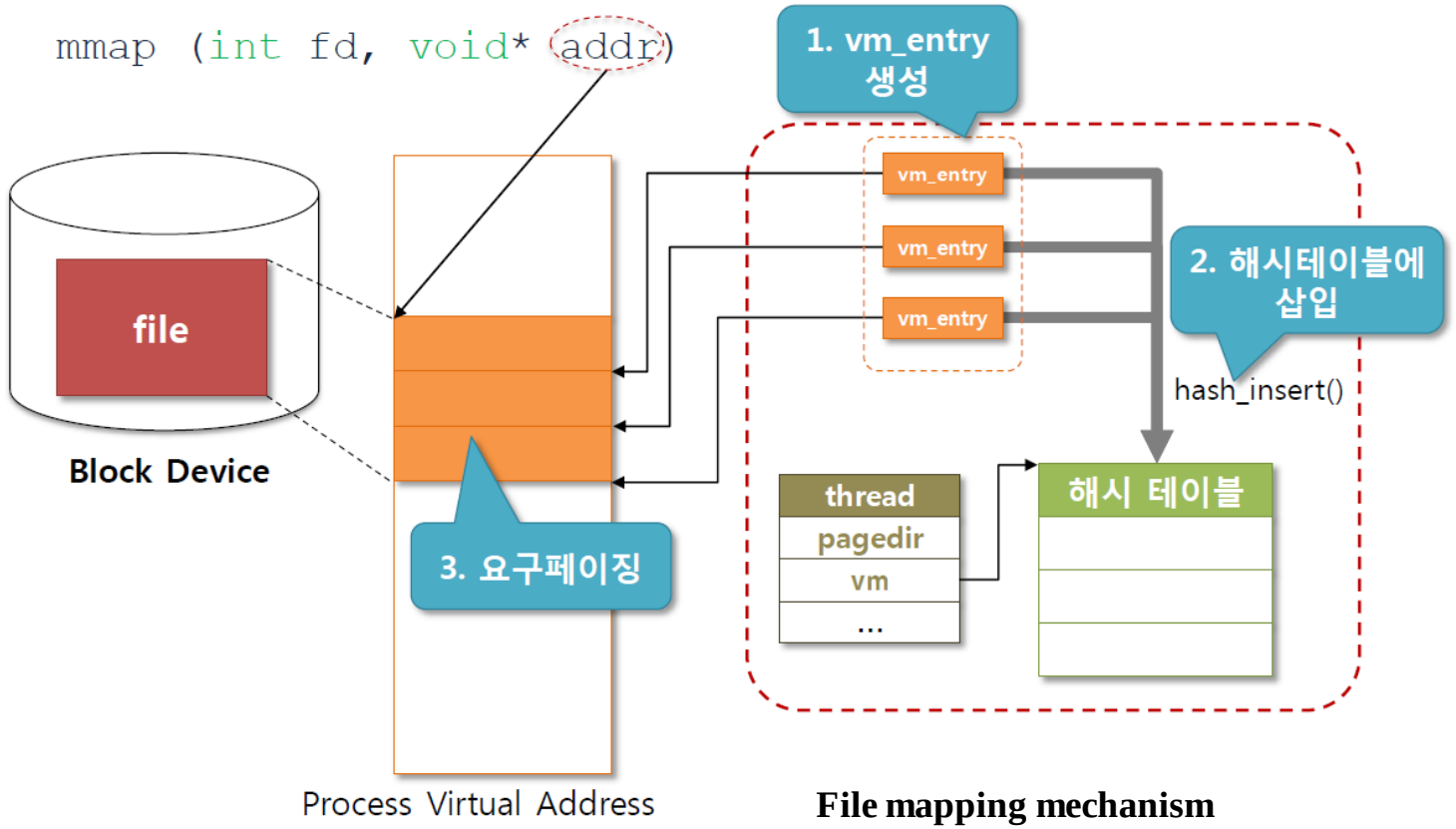


Problem 3.

- **Memory Mapped Files (required, 1pt)**
- **Requirement:**
  - Implement follow system calls.
    - Mapid_t mmap (int fd, void *addr)
    - Void munmap (mapid_t mapping)
  - And Implement memory mapped files.

Problem 3.

- **Memory Mapped Files (required, 1pt)**
- Problem Manual: Current state
  - Current memory mapped pages are file-backed mapping.
  - When the page fault occurs, data in page allocated to frame immediately.
  - So we should implement mmap and munmap.

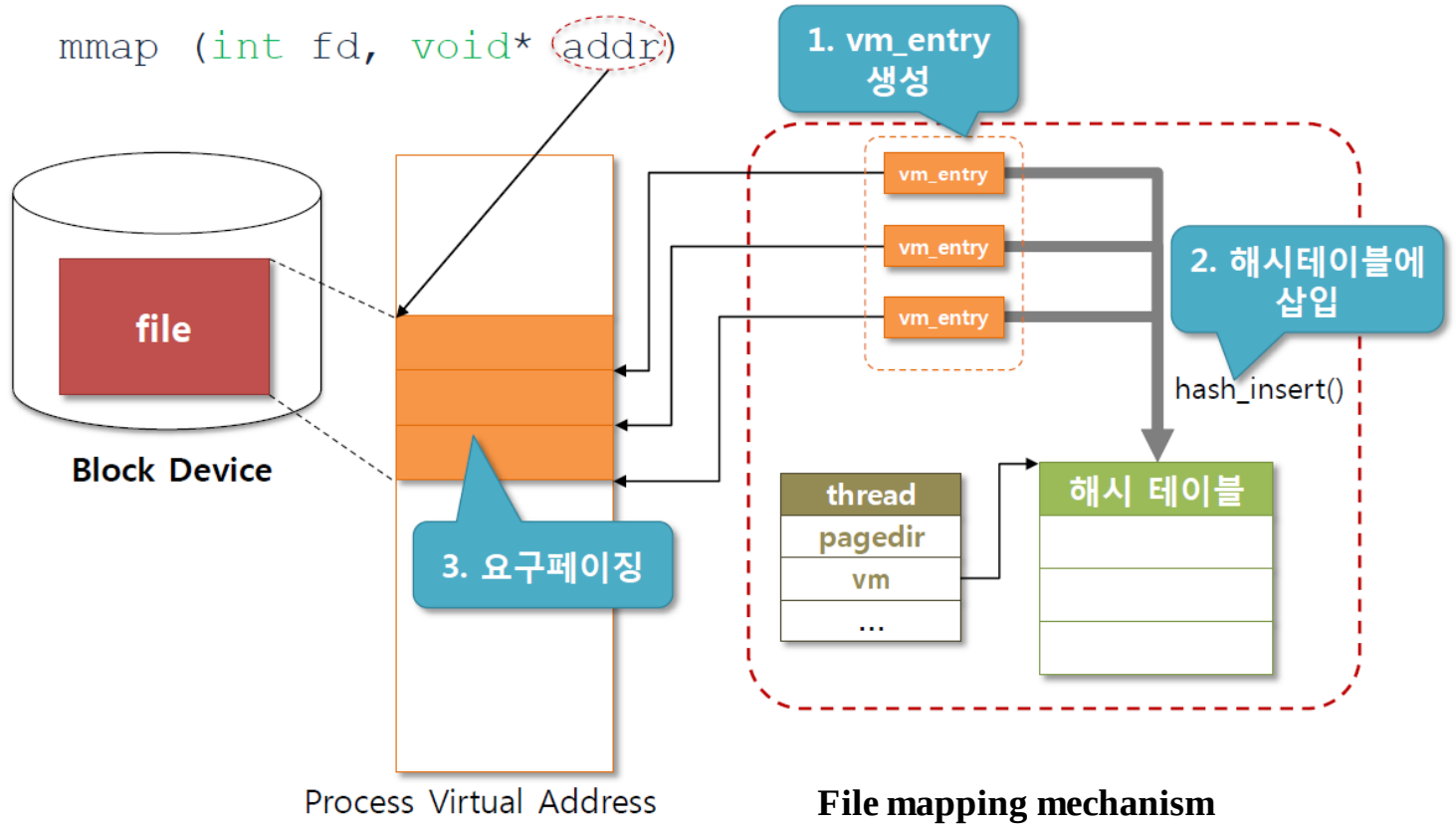


File mapping mechanism

Problem 3.

- **Memory Mapped Files (required, 1pt)**
- Problem Manual:
  - mmap: maps the file open as fd into the process's virtual address space.
  - Must lazily load pages in mmap regions and use the mmaped file itself as backing store for the mapping.
  - If successful, returns a "mapping ID".
  - If not, must return -1.

  - munmap: Unmaps the mapping designated by mapping, which must be a mapping ID returned by a previous call to mmap by the same process that has not yet been unmapped.



File mapping mechanism

Problem 4.

- **Accessing User Memory (required, 1pt)**
- **Requirement:**
  - You will need to adapt your code to access user memory while handling a system call.
  - While accessing user memory, your kernel must either be prepared to handle such page faults, or it must prevent them from occurring.

Problem 4.

- **Accessing User Memory (required, 1pt)**
- Problem Manual: Current state
  - If kernel code accesses non-resident user pages, page fault will result.
  - So while you accessing to user memory, kernel must either be prepared to handle such page faults, or it must prevent them from occurring.
  - Preventing such page faults requires cooperation between the code within which the access occurs and your page eviction code. (Such as “pinning” or “locking”.)

6. Report

- Problem, Design, and Implementation (required, 1pt)
- Please submit a report (limit 5 pages, Korean possible, no explicit demerits on exceeding page limits) that includes:
  - Student IDs, names, and team number
  - Each problem should be described in the form of
    - Problem definition
    - Algorithm design
    - Implementation (what you have added to solve the equation and corresponding file names)

7. What to submit?

- Submit a report (5pages limit) to TA via email (minwook-lee@gist.ac.kr)and upload implemented source code to your github repository.
- The format of report does not matter if it clearly shows how you implemented the project. (please clarify which file of your code has been changed)

- Reference:
  1. https://web.stanford.edu/class/archive/cs/cs140/cs140.1088/projects/pintos/pintos_4.html