Virtual Memory Manager

1. Base Design

- We first set mprotect (PROT_NONE) for all the virtual pages, so that whenever they are accessed while doing either a write operation or a read operation, a page fault occurs, and segfault_handler is called.
- The allocator keeps its own record of "resident pages" in RAM.
- FIFO: Created a singly linked list of FIFOPhysicalFrame nodes.
- Third Chance: Created a circular list of ThirdChanceFrame nodes.
- mm_init() initializes setupConfig, which holds information provided from main.c.
- mm_finish() frees FIFO list nodes when policy is FIFO, and frees the circular list when policy is Third Chance.

2.1 Common Architecture

- mprotect(vm, vmsize, PROTNONE) protects the whole virtual page region; any access raises SIGSEGV.
- The handler for SIGSEGV is triggered with sigaction(SIGSEGV).
- Inside the handler, we get the address at which the page fault occurred. We then calculate the virtual page number of the address by dividing the virtual address into equal partitions of virtual page sizes.
- For the 2nd bit of the error register (REG_ERR), we check if the bit is set or not. If the bit is set, then a page fault occurred on a write operation; otherwise, in a read operation.

2.2 FIFO (First-In-First-Out) Replacement:

- A singly linked list of FIFOPhysicalFrame nodes is used.
- Each node contains frame_number, virtualPageNumber, isWritePerformed, and next pointer, which points to the next node.
- We maintained the isWritePerformed boolean value so that when we are evicting a virtual page, we can understand if writeback needs to be performed.
- We first check if the virtual page is already present in physical memory.
- <u>Hit (page already present)</u>:
  - If it's a write fault (fault type = 2) on a resident which only has read permissions, we mark write performed so that writeback can be performed and update the mprotect (PROT_READ | PROT_WRITE).
- <u>Miss (page not present)</u>:
  - If physical frames are full (countLinkedListNodes( ) == numframes), we evict the oldest virtual page (present at the head node). We set mprotect (PROT_NONE) because we want the faults to happen again on this evicted page.
  - Then, at the last position in the linked list but at the same physical frame, we add a new virtual page. If this page is added to physical memory while doing a write operation, we provide PROT_READ | PROT_WRITE permissions, and for read operations, we provide PROT_READ permissions.
- We did log accordingly.

2.3 Third Chance Replacement:

- The structure, circular linked list, ThirdChanceFrame holds framenumber, virtualPageNumber, R (reference bit), M (modified bit), passcount, and next pointer.
- A global pointer, thirdChanceHead, acts as the clock hand mentioned in the project description.
- For insertion in the circular linked list, we add at the last of the linked list before the head.
- Hit (page already present):
    - If the page already has read permissions, and a write operation is performed, its fault type is 2.
    - If the page was modified and it went under the clock hand, it will have mprotect(PROT_NONE) permissions. In this case:
        1. If a write operation caused a fault, fault type is 4, and we need to again provide the mprotect(PROT_READ | PROT_WRITE) permissions to it, and its pass counts are reset to give it three chances.
        2. If a read operation caused a fault, fault type is 3, and we need to again provide the mprotect(PROT_READ) permissions to it, and its pass counts are reset to give it three chances.
- Miss (page not present):
    - Like FIFO, if physical frames are full, we need to perform eviction. Below is how we wrote a helper function for finding an eviction candidate (node from a circular linked list).
    - **Eviction selection rule**: findEvictionNode() repeatedly advances thirdChanceHead and applies these rules:
        1. (R=0, M=0) → immediately return as a victim.
        2. (R=1, M=0) → clear R=0, give one more chance, set the page to PROTNONE, advance.
        3. (R=1, M=1) → clear R=0, set passcount=1, has 2 more chances, set the page to PROTNONE, advance.
        4. (R=0, M=1) → increment passcount from 0→1→2; only when it's already past does it finally return the node as victim. Basically, in this case candidate gets 3 chances.
    - We then insert the new virtual page at the same physical frame, but at the last of the circular linked list.
    - Provide permissions according to write and read operations, causing the page fault.

3. Key Implementation Choices

- We have used protection-based paging simulation, by using mprotect (PROTNONE / PROTREAD / PROTREAD PROTWRITE) as the mechanism to trigger and control faults.
- We also implemented write-fault detection using ucontext error code (REGERR & 0b010), so decisions differ for read vs write faults.
- We made sure that there are separate metadata per policy: FIFO uses isWritePerformed as the dirty indicator, Third Chance uses R, M, and passcount to implement multi-pass eviction.

- We initially only thought about implementing FIFO replacement, therefore used a singly linked list. While implementing the third replacement policy, we created a separate circular linked list, as we didn’t want to touch the implementation of FIFO.
- We implemented frame reuse on eviction, such that the victim's frame number becomes the incoming page’s frame-number (both FIFO and Third Chance).
- Required logging pipeline, such that the handler computes (virtpage, faulttype, evictedpage, writeback, phyaddr) and records it via mmlogger into mmstats.

4. Known Limitations / Not Implemented

As much as we know, we are not aware of any limitations. We tried to cover most of the edge cases.

5. Testing Summary

- We tried running all the inputs given in the GitHub repo for both FIFO and third replacement. All the cases are passing, and we also tried running multiple times for all inputs. The outputs are consistent on Lab 135 machines.
- Also, we tried to run inputs from the project details, and we were successful in running all the edge cases.