Memory Management

Parikshit Gehlaut	Vemula Chandrahaas Reddy	Soumyadeep Das
220150009	220010062	220010056

November 11, 2024

1 Introduction

In this assignment, we developed a Virtual Memory Manager simulator in C++ that explores various aspects of memory management. Specifically, we implemented a simulator that experiments with multiple page replacement policies and allocation methods. The command-line inputs to our simulator include page size, number of memory frames, replacement policy, allocation policy, and a path to a memory trace file. We conducted experiments with various page sizes, number of memory frames, replacement policies (Optimal, FIFO, LRU, Random), and allocation policies (Global and Local).

2 Implementation Approach

- **Process Class**
  - The Process class manages a process's virtual memory by tracking its process id, page table,local page faults, and frame allocations under various replacement policies (**FIFO, RANDOM, LRU, OPTIMAL**).
  - Implemented a `PageTable` using a unordered map, key - page number,value - frame number, instantiated four times for each process. The unordered map allows for fast and efficient lookups for page mappings, helping in quick access of virtual to physical address translation.
- **MemoryManager Class**
  - Functions as OS, contains all essential methods that help us to solve the assignment.
  - Checks PageTable, allocates Frames, checks for free Frames, according to different replacement policies and allocation policies.
- **Miscellaneous**
  - Physical Memory - Implemented using C++ maps, key - frame number , value - process id, page number pair.
  - globalPageFault - Maintained as global variable that keeps track of total faults by all processes.
- **General Flow and Page Fault Handling**
  - For each memory access, the virtual address is checked within the relevant process's page table, using page number as key using *checkPageTable()* funtion ,thats return if mapping found found after updating metadata.

- If no mapping found, a page fault occurs, the fault counters (globalPageFault and per-process[localPageFault]) increment, which then calls *checkFrame()* function, which checks if free frames present or not.
- And allocates frame, either by finding an empty one or by applying the replacement policy, taking into account allocation policy.

- **Allocation Policies**
  - Implemented Global and Local allocation policies. In Local, the frames are evenly divided among processes.

- **Replacement Policies**
  - Implemented four policies: Optimal, FIFO, LRU, and Random. Each policy influences frame allocation differently depending upon the allocation policy.
  - **Global**
    1. **FIFO** : Implemented using a global queue fifoQueue which keeps track of the present allocated frames, pushing the latest allocated frame to the bottom of the queue, replacing frame at start when replacement policy called.
    2. **LRU** : Made a global list lrulist which keeps track of the allocated frames, pushing back a frame to the bottom of the list if in case its a hit, or poping the head of the linked list in case its a page fault.Always ensuring lru frame is at start of linked list.
    3. **Random** : Used a C++ inbuilt function *rand* to generate a random index in the range of number of frames, in case of replacement evicting any random frame based of the randomly generated number from the page table.
    4. **OPTIMAL** : Created FrameToBeRemoved() function that selects frame with further next usage . It first maps each page's next occurrence index after the current index, CurrIdx. Then, it checks each frame in frameVec to determine how far in the future it will be used next. The frame with the maximum distance to its next usage is chosen for replacement.
  - **Local**
    1. **FIFO** : Principle is the same as global fifo but here we instead used a local variable $allocatedFrames$ defined for each process, which seperately keeps tab on which frames were allocated to each process.
    2. **LRU** : Same as global LRU, used another local variable $lallocatedFrames$ to keep track of allocated frames.
    3. **Random** : Here we used local variable $rallocatedFrames$ to keep track of allocated frames for each process, and only evicting a random frame from this list in case of page fault for respective process.
    4. **OPTIMAL** : Created FrameToBeRemovedLocal() function that selects frame specific to that process that are in particular vector (oallocatedFrames) with further next usage . It first maps each page's next occurrence index after the current index, CurrIdx. Then, it checks each frame in oallocatedFrames to determine how far in the future it will be used

next. The frame with the maximum distance to its next usage is chosen for replacement.

3 Experiments and Observations

The performance was evaluated by observing the number of page faults under various parameter combinations. We tested different page sizes, frame numbers, and policies across two memory trace files (a small trace for testing and debugging and a larger one for analysis).

3.1 Experiment Setup

- Page sizes ranged from 1 KB to 8 KB.
- Memory frames varied between 16 and 2048 frames.
- The replacement policies tested were Optimal, FIFO, LRU, and Random.
- Allocation policies tested were Global and Local.

3.2 Results

The following table summarizes the number of page faults for different configurations:

Policy	**Page Size (KB)**	**No of Frames**	**Global**	**Local**
Optimal	4096	512	2184	4490
FIFO	4096	512	4591	6999
LRU	4096	512	4082	6892
Random	4096	512	4703	7115

More complex analysis provided below using graphs

3.3 Analysis

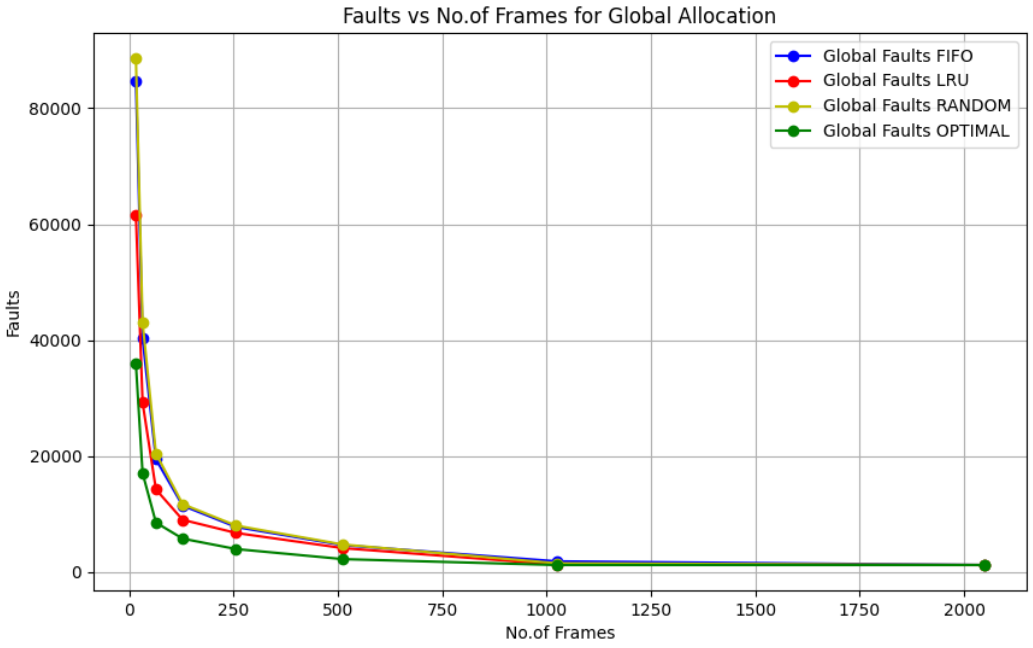

Figure 1: Faults vs No.of Frames for Global Allocation fixing page size 4KB

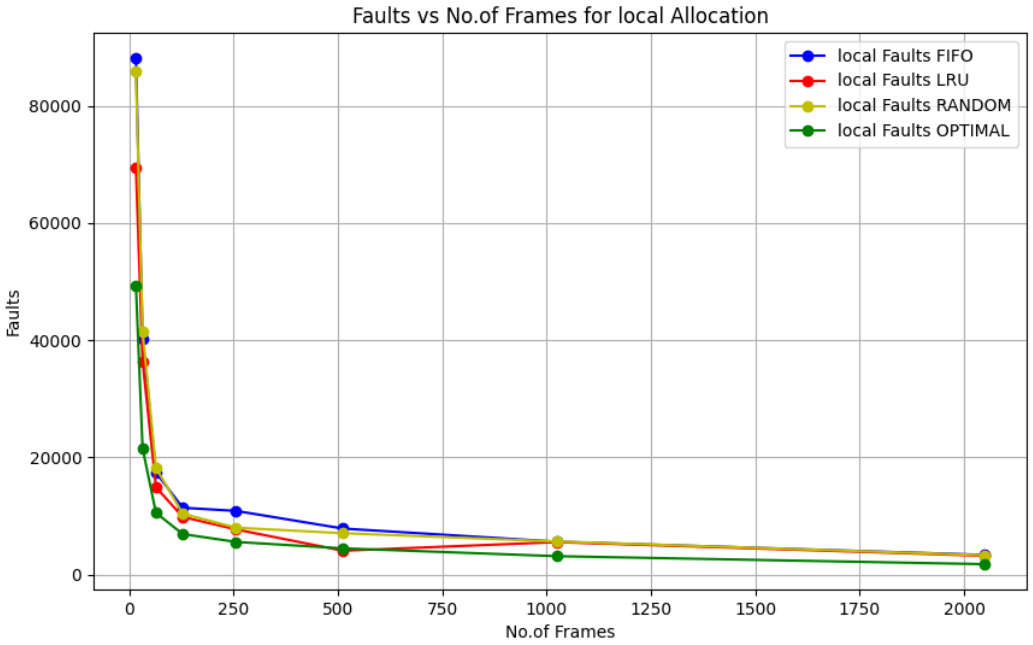

Figure 2: Faults vs No.of Frames for Local Allocation fixing page size 4KB

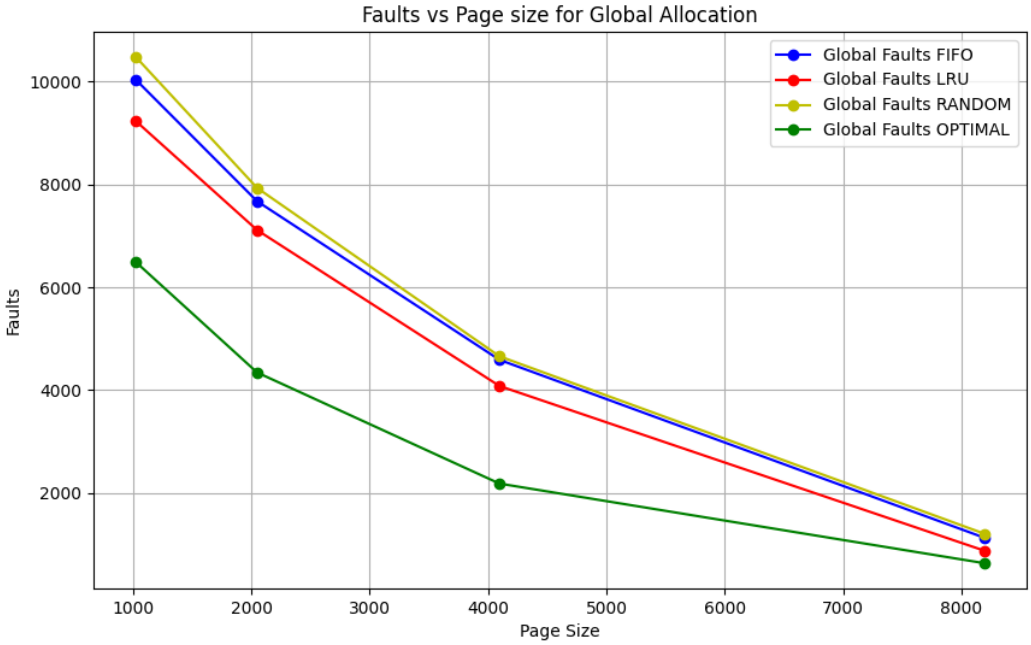

Figure 3: Faults vs Page size for Global Allocation fixing no.of frames at 512

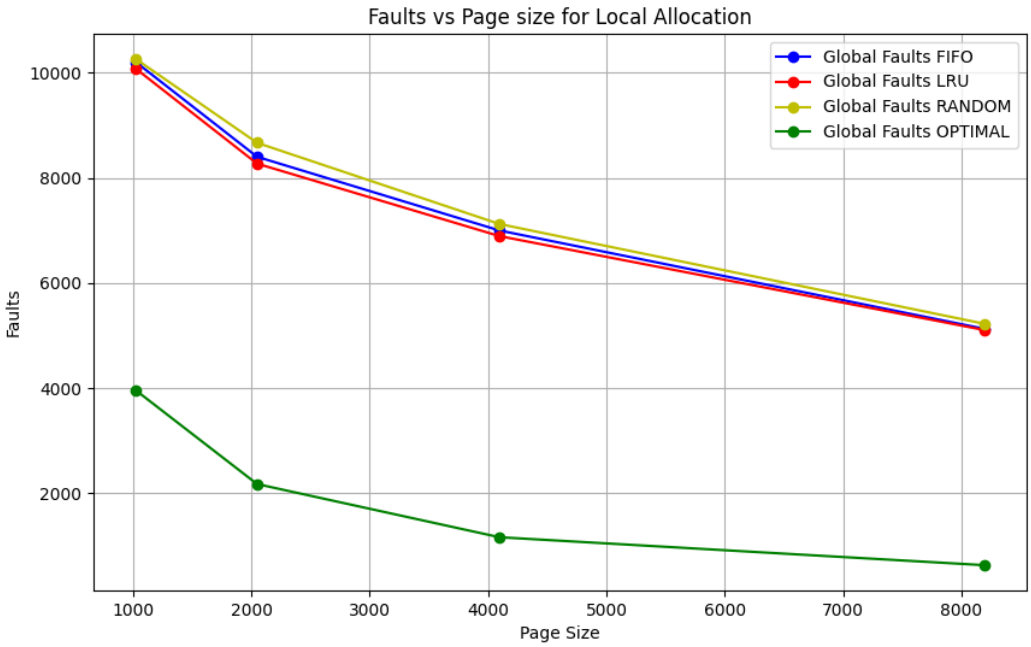

Figure 4: Faults vs Page size for Local Allocation fixing no.of frames at 512

4 Conclusion

Through this assignment,

- we observed that replacement policies impact page fault rates significantly, with Optimal set as benchmark and LRU outperforming FIFO and Random. Global allocation tends to minimize page faults across processes.
- Further, we also observed that as we increase page size or number of frames faults reduces noticeably.
- Through this assignment we got a keen understanding of the importance of selecting suitable page size, number of frames, page replacement and allocation strategies to balance system performance and process demands.