Name : Rafael
Student ID : 32215130
Major : Global Software Convergence
Freedays : 0

1. Introduction

Virtual memory is a cornerstone of modern operating systems, enabling efficient utilization of physical memory while providing isolation and flexibility to processes. This project simulates a multi-process environment where demand paging, swapping, and other advanced memory management techniques are implemented. The simulation incorporates detailed logging for debugging and evaluation.

---

2. Objectives

- To understand the concept of virtual memory and address translation with paging.
- To simulate demand paging and efficient page table management.
- To implement advanced memory management features like swapping and detailed logging.
- To gain hands-on experience with system-level programming and debugging.

---

3. Implementation Details

Key Components

1. **Page Tables**:
    - Each process is assigned a page table for virtual-to-physical address mapping.
    - Entries in the page table include the frame number and validity flags.
2. **Demand Paging**:
    - Pages are allocated on demand, with faults logged when unmapped pages are accessed.
3. **Swapping**:
    - When memory is full, the Least Recently Used (LRU) algorithm is employed to evict pages.
    - Evicted pages are stored in a simulated disk file for later retrieval.
4. **Logging**:

    - Logs are generated for every memory access, page fault, and swap operation.

Compilation and Execution

1. Save the source code as `memory_management_simulation.c`.
2. Compile using:
   gcc -o memory_management_simulation memory_management_simulation.c -lpthread
3. Run the executable:
   ./memory_management_simulation
4. Check the logs generated in `simulation_logs.txt` for a detailed breakdown of operations.

---

4. Challenges and Learning Experiences

Challenges:

- **Understanding Paging and Address Translation**: Translating virtual addresses into physical addresses required careful manipulation of bitwise operations and understanding of memory architecture.
- **Implementing Swapping**: Evicting pages efficiently while ensuring data integrity during page-in and page-out operations was non-trivial.
- **Concurrency Issues**: Synchronizing memory access and page table updates across multiple simulated processes was complex.

Learning Experiences:

- **In-depth Understanding of Virtual Memory**: The project reinforced theoretical knowledge of virtual memory by implementing concepts like demand paging and LRU swapping.
- **Debugging Complex Systems**: Managing multiple interdependent components emphasized the importance of modular programming and detailed logging.

---

5. Unique Features of the Code

1. **Detailed Logging**:
    - Comprehensive logs include page faults, memory utilization, and swap operations.
2. **LRU Swapping**:
    - A custom implementation of the LRU algorithm efficiently manages page replacements.
3. **Scalability**:
    - The simulation handles up to 10 processes concurrently, demonstrating realistic multi-process memory management.

4. **Code Modularization**:
    - Functions like `handle_page_fault()` and `swap_out()` isolate key functionalities, improving readability and maintainability.

---

6. Possible Future Enhancements

- **Two-Level Paging**: Implement a hierarchical page table structure to optimize memory usage further.
- **Advanced Page Replacement Policies**: Explore algorithms like Clock or LFU for page replacement.
- **Enhanced Visualization**: Generate graphical representations of memory utilization and page table states.
- **Dynamic Process Creation**: Add support for processes to be created dynamically during the simulation.

---

7. Conclusion

This project provided a comprehensive understanding of virtual memory management and paging mechanisms. By simulating demand paging, swapping, and concurrency, we bridged the gap between theory and practice in operating system concepts. The detailed logs serve as a valuable tool for debugging and evaluating the performance of the implementation.

---

8. How ChatGPT Helped

Assistance Highlights:

1. **Understanding Complex Concepts**:
    - ChatGPT clarified the nuances of demand paging and swapping, particularly the interplay between page tables, MMU, and memory.
2. **Optimized Code Snippets**:

Suggestions for modularizing code, e.g.,

```
int frameALLOCATE() {
    if (FREEframelist.size == 0) {
        return -1; // if no free frames available
    }
    int frame_number = FREEframelist.frames[--FREEframelist.size];
    return frame_number;
}
```

This modular approach improved both clarity and maintainability.

3. **Debugging and Logging**:

Enhanced the logging system to provide detailed breakdowns of memory operations, as seen in the logs:

```
[Tick 49] Process 4 accessed VA 0xA8E63
  -> VA Breakdown: Page Number = 0xA8, Offset = 0xE63
  -> No valid mapping in the page table. PAGE FAULT !
  -> Memory Full! Swapping required.
    -> Swapping out Page no. 222 of Process 1 (Frame 14).
    -> Saving Page 222 of Process 1 to disk (evicting Frame 14).
    -> Page Table Updated: Process 4, VA Page 168 -> Frame 14
```

Algorithm Suggestions:
Provided insights into implementing the LRU swapping mechanism:

```
int victimframeFIND() {
    int oldest_tick = tickCURRENT;
    int victim_frame = -1;
    for (int i = 0; i < NUM_FRAMES; i++) {
        for (int p = 0; p < NUM_PROCESSES; p++) {
            for (int j = 0; j < PAGE_TABLE_SIZE; j++) {
                if (process[p].page_table[j].valid && process[p].page_table[j].frame_number == i) {
                    if (process[p].page_table[j].last_used < oldest_tick) {
                        oldest_tick = process[p].page_table[j].last_used;
                        victim_frame = i;
                    }
                }
            }
        }
    }
    return victim_frame;
}
```

4. **Error Resolution**:
    - Helped resolve compilation issues and suggested improvements for efficient memory management.
