CSE 312/504 – Operating Systems

Homework 2

1710044005 - Umit Altintas

Introduction

In this project, we have designed and implemented a simulated and simplified virtual memory management system using C++. The system includes a number of page replacement algorithms such as Second-Chance (SC), Least-Recently-Used (LRU), and Working Set Clock (WSClock). The goal of this project is to perform integer array arithmetic operations and search algorithms on a virtual memory space, with the ability to store overflow data on a disk file.

System Overview

The simulated virtual memory management system consists of several key components: MMU (Memory Management Unit), Page Table, and Page Table Entry. The MMU handles memory access and page replacement, while the Page Table stores the mapping between virtual and physical memory addresses. Each entry in the Page Table represents a single page and contains information about its presence in physical memory, modification status, reference counter, and physical address.

Page Table Entry Structure

The PageTableEntry struct represents each entry in the page table. It contains the following fields:

- `isEntryPresent` : Indicates whether the page is present in physical memory (0 or 1).
- `isEntryModified` : Indicates whether the page has been modified (0 or 1).
- `referenceCounter` : Keeps track of the reference count for the page.
- `physicalAddress` : Stores the physical address where the page is located.

Page Table Structure

The PageTable struct represents the entire page table and includes the following fields:

- `virtualAddressSpaceSize` : Size of the virtual address space.
- `physicalMemorySize` : Size of the physical memory.

- `pageTableSize` : Number of entries in the page table.
- `pageTableEntries` : Vector to store all the page table entries.
- `lastAccessTime` : Keeps track of the last access time for page replacement algorithms.

MMU Class

The MMU class is responsible for handling memory access and page replacement operations. It has the following public member functions:

- `get(unsigned int index)` : Retrieves the value at the specified index in the virtual memory.
- `set(unsigned int index, int value)` : Sets the value at the specified index in the virtual memory.
- `printStats()` : Prints the statistics related to memory access and page replacement.

The private member functions of the MMU class include:

- `init()` : Initializes the MMU object by initializing the page table, virtual memory, and physical memory.
- `initPageTable()` : Initializes the page table by allocating memory for page table entries.
- `initVirtualMemory()` : Initializes the virtual memory by creating a disk file and resizing it based on the page table size.
- `initPhysicalMemory()` : Initializes the physical memory by allocating memory for the physical memory array.
- `getPhysicalFrameFromVirtual(uint32_t virtualFrame)` : Retrieves the physical frame corresponding to the given virtual frame.
- `findEmptyMemoryFrame()` : Finds an empty memory frame in the physical memory.
- `handlePageFaultForVirtualFrame(uint32_t virtualFrame)` : Handles a page fault for the given virtual frame by replacing a page in physical memory.
- `setPage(uint32_t vFrame, uint32_t pFrame)` : Sets the page table entry for the virtual frame to point to the specified physical frame.
- `isTimeThresholdReached()` : Checks if the time threshold for page replacement has been reached.
- `isMemoryFull()` : Checks if the physical memory is full.
- `isFrameOccupied(uint32_t frameIndex)` : Checks if the given frame index in physical memory is occupied.
- `updateAccessTime()` : Updates the last access time for page replacement algorithms.
- Page replacement algorithms:
  - `lruAlgo()` : Implements the Least Recently Used (LR

U) page replacement algorithm.

- `scAlgo()` : Implements the Second-Chance (SC) page replacement algorithm.
- `wscAlgo()` : Implements the Working Set Clock (WSClock) page replacement algorithm.

MMU_Config Class

The MMU_Config class represents the configuration settings for the MMU. It includes the following fields:

- `frameSize` : Size of each memory frame.
- `numPhysical` : Number of frames in physical memory.
- `numVirtual` : Number of frames in virtual memory.
- `algo` : The selected page replacement algorithm (SC, LRU, WSC, or NONE).
- `path` : Path to the disk file for storing overflow data.

MMU_Statistics Class

The MMU_Statistics class keeps track of various statistics related to memory access and page replacement. It includes the following fields:

- `getAccessCount` : Number of "get" operations performed.
- `setAccessCount` : Number of "set" operations performed.
- `pageMisses` : Number of page faults that occurred.
- `pageReplacements` : Number of page replacements that occurred.
- `readCount` : Number of read operations performed.
- `writeCount` : Number of write operations performed.

Implementation Details

The implementation code provided includes the necessary implementations for the MMU class, MMU_Config class, and MMU_Statistics class. The code defines helper functions and variables for handling page replacement algorithms. Here are some important implementation details:

- The MMU constructor initializes the MMU object based on the provided configuration. It selects the appropriate page replacement algorithm based on the configuration settings.
- The MMU destructor cleans up the allocated resources, closes the disk file, and clears the page table entries and WSClock list.
- The `get` and `set` functions of the MMU class perform memory read and write operations, respectively. They handle page faults and page replacements as needed.
- The `getPhysicalFrameFromVirtual` function retrieves the physical frame corresponding to a virtual frame. If the page is not present in physical memory, it triggers a page fault and selects a victim page for replacement.
- The `handlePageFaultForVirtualFrame` function handles a page fault for the given virtual frame. It selects a victim page based on the chosen page replacement algorithm and updates the page table and physical memory accordingly.
- The `setPage` function sets the page table entry for a virtual frame to point to the specified physical frame. It reads the data from the disk file and updates the page table entry.
- The page replacement algorithms (LRU, SC, and WSClock) select a victim page based on different criteria, such as reference counters, modification status, and time thresholds.
- The `printStats` function prints the statistics related to memory access and page replacement.

Main Function The main function serves as the entry point for the program. It performs the following steps:

Parses the command-line arguments to obtain the configuration settings for the MMU. Creates an instance of the MMU class using the provided configuration. Fills the virtual memory with random integers. Creates multiple threads to perform matrix-vector multiplication, vector-vector multiplication, array summation, linear search, and binary search operations on the virtual memory. Joins the threads and waits for them to finish their tasks. Prints the statistics of memory access and page replacement. Conclusion In this project, we have successfully implemented a simulated virtual memory management system in C++. The system incorporates page replacement algorithms and supports integer array arithmetic operations and search algorithms on a virtual memory space. The design and implementation of the MMU class, Page Table, and Page Table Entry provide a solid foundation for efficient memory management and page replacement.

The provided code includes the necessary functionality for the simulated virtual memory management system. However, certain parts, such as the matrix-vector multiplication and array summation algorithms, as well as the search algorithms, are missing. These components need to be implemented based on the project requirements.

It is recommended to thoroughly test the system and evaluate its performance using different configurations and workload scenarios. This will ensure the correctness and efficiency of the implemented system.

Overall, this project has provided valuable insights into virtual memory management systems, page replacement algorithms, and their practical implementation.