metin, logo, grafik, grafik tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Gebze Technical University**

Computer Engineering - CSE 312 Operating Systems

Virtual Memory Systems and Paging

Homework 1 Report

BUKET GENCER

210104004298

**INTRODUCTION**

The main goal of this project is to design and implement a simple virtual memory management system in C/C++, focusing on page replacement algorithms. The system simulates how an operating system handles virtual memory, with a particular focus on managing memory when the virtual memory exceeds the available physical memory. In this project, I used a single thread to fill, sort, and search a large virtual memory array, which simulates real-world memory operations. Although the project originally required using two threads, I faced an issue that I couldn't fully debug, so I completed the assignment with one thread instead.

**Design Overview**

The virtual memory management system is designed to efficiently handle memory access using a combination of page tables, physical memory, and virtual memory. The page table serves as a critical component, mapping virtual addresses to physical memory locations. Each page table entry includes information such as the frame number, valid bit, modified bit, referenced bit, and the last access time, enabling the system to manage memory efficiently.

Physical memory is represented as an array that holds the actual data in frames, with each frame corresponding to a page in the virtual memory. Virtual memory, which is larger than the physical memory, stores data that can be swapped in and out of physical memory as needed.

Two page replacement algorithms, Least Recently Used (LRU) and Clock (CL), are implemented to manage situations where a page needs to be replaced in physical memory. These algorithms ensure that the system optimizes memory usage based on access patterns.

A diagram illustrating the design of this system will be included to provide a visual representation of the relationships between these components.

The system is designed to handle two threads, each with its own virtual memory and page table, while sharing a single physical memory. The physical memory is global and accessible by both threads, allowing them to store and retrieve data as needed.

In this design, the disk is divided into two local sections, one for each thread. The first half of the disk is allocated for the first thread, and the second half is reserved for the second thread. This separation ensures that each thread has its own dedicated space for storing pages that cannot fit into the physical memory, preventing interference between the threads' data.

metin, el yazısı, çizim, yazı tahtası içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Page Table Entry Structure**

The PageTableEntry struct represents each entry in the page table and includes the following fields:

* **frameNumber**: This field indicates the frame number in physical memory where the page is currently stored. If the page is not in physical memory, this field is set to -1.
* **valid**: A valid bit that indicates whether the page is currently in physical memory (1 if it is, 0 otherwise).
* **modified**: This bit indicates if the page has been modified since it was loaded into physical memory. It is used to determine if the page needs to be written back to the disk before being replaced.
* **referenced**: This bit is used by the page replacement algorithms to track whether the page has been accessed recently. It helps in determining which page to replace.
* **lastAccessTime**: This field records the last access time of the page in nanoseconds, which is particularly useful for implementing the Least Recently Used (LRU) page replacement algorithm.

Here's the code snippet for the PageTableEntry struct:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Page Table Structure**

The page table is a key component of the virtual memory system, storing the mapping between virtual pages and physical frames. Each thread in the system has its own page table, which contains PageTableEntry structures for each virtual page. The page table is initialized with each entry set to indicate that no pages are currently in physical memory.

The page table is used to translate virtual addresses into physical addresses. When a thread tries to access a page, the system checks the page table to see if the page is in physical memory. If it is, the physical address is calculated using the frame number. If not, a page fault occurs, and a page replacement algorithm is triggered to load the page into memory.

Here’s how the page table is initialized:

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Virtual Memory Structure**

Virtual memory is organized into pages, each of which can be mapped to a frame in physical memory. In this system, each thread has its own virtual memory, which is larger than the available physical memory. The virtual memory is divided into equal-sized pages, and these pages are mapped to physical frames via the page table.

When a thread tries to access data, it provides a virtual address, which is split into a page number and an offset. The page number is used to look up the frame number in the page table, and the offset is used to locate the exact data within the frame.

Here’s how virtual memory is filled with random data:

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Physical Memory Structure**

Physical memory is organized into frames, each of which can hold one page of data. In this system, physical memory is shared between the two threads, and each frame can belong to any thread. When a page needs to be loaded into physical memory, an empty frame is used if available; otherwise, a page replacement algorithm (LRU or Clock) selects a victim page to be replaced.

The physical memory is managed by maintaining a vector of physicalMemoryEntry structs, which store the actual data, the owning thread, and the disk index for data that has been swapped out.

Here’s how physical memory is initialized:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Disk Structure**

In the virtual memory management system, the disk is used as a secondary storage to hold data that cannot fit into the physical memory. The disk is represented as a file in the system, and it is divided into two parts: the first half is dedicated to thread 1, and the second half is reserved for thread 2. This separation ensures that each thread has its own space on the disk to store its pages when they are swapped out of physical memory.

The disk file is initialized with placeholder values (e.g., -1), and each time a page needs to be evicted from physical memory, it is written to the appropriate section of the disk based on the thread it belongs to. The initializeDisk() function is responsible for setting up this structure by creating the disk file and populating it with initial values.

Relevant code snippet:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**get and set Functions**

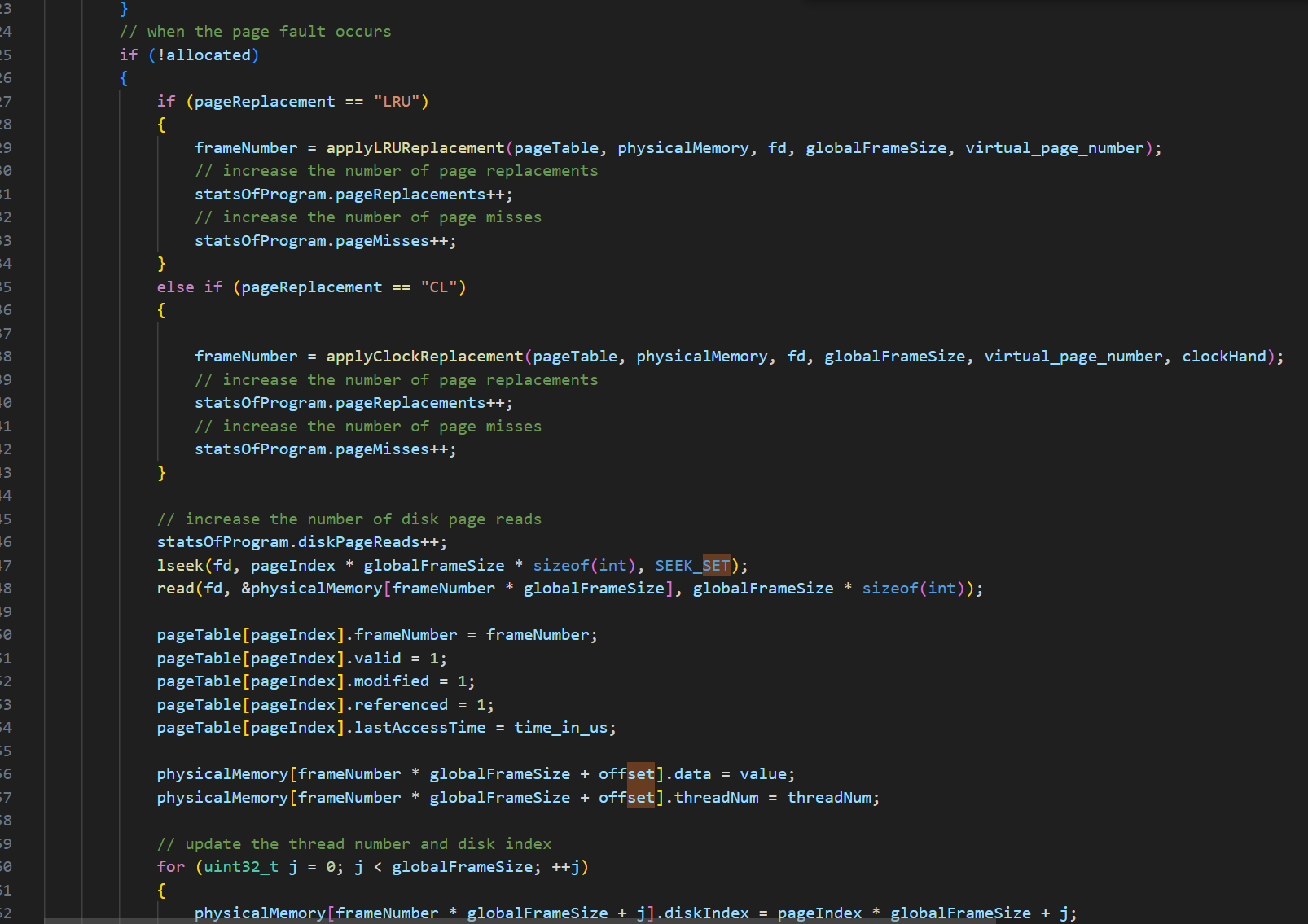
The get and set functions are the core components of the virtual memory management system. They are used to access and modify the data in virtual memory, respectively. These functions ensure that when a thread tries to read or write data, the data is correctly mapped between virtual memory and physical memory, using the page table.

* **set Function**: This function writes a value to a specific index in the virtual memory. If the page containing that index is not already in physical memory, the page replacement algorithm is triggered, and the page is loaded from the disk if necessary. The set function also updates the page table with relevant information such as the modified bit, the frame number, and the last access time.
* **get Function**: This function retrieves the value from a specific index in the virtual memory. Similar to the set function, if the page is not in physical memory, the page replacement algorithm is triggered to load the page into memory. The get function updates the page table with information like the referenced bit and the last access time.

Relevant code snippets:

metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu



metin, elektronik donanım, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

These functions are critical for managing the virtual memory efficiently and ensuring that the system correctly handles memory accesses by using the page table to map virtual addresses to physical memory or the disk when necessary.

**Page Replacement Algorithms**

**Least Recently Used (LRU)**

The Least Recently Used (LRU) algorithm is a page replacement strategy that selects the page that has not been used for the longest period of time for replacement. The assumption behind LRU is that pages that have been used recently will likely be used again soon, while pages that have not been used for a while are less likely to be needed in the near future.

**How LRU Works:**

* When a page needs to be loaded into physical memory and all frames are occupied, the LRU algorithm checks the "last access time" of all pages currently in physical memory.
* The page with the oldest access time is selected for eviction.
* The evicted page is written to the disk if it has been modified, and the new page is loaded into the freed frame.

**LRU Implementation in the System:** In this system, the LRU algorithm is implemented by maintaining a lastAccessTime field for each page in the page table. This field records the last time the page was accessed (either read or written). During the page replacement process, the system scans all the pages in physical memory to find the one with the oldest lastAccessTime. This page is then evicted to make room for the new page.

**Code Snippet:**

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Clock (CL)**

The Clock algorithm is an approximation of the LRU algorithm. It uses a circular list of pages (resembling a clock) and a pointer that moves through the pages. Each page has a "referenced" bit that indicates if the page has been accessed recently. The Clock algorithm is also known as the "Second Chance" algorithm because it gives each page a second chance before eviction.

**How Clock Works:**

* The Clock algorithm maintains a circular list of pages with a pointer (often referred to as the "clock hand").
* When a page needs to be replaced, the algorithm checks the page where the clock hand is pointing.
* If the referenced bit of the page is 0, the page is evicted. If the bit is 1, the algorithm clears the bit and moves the clock hand to the next page.
* This process continues until a page with a 0 referenced bit is found.

**Clock Implementation in the System:** In this system, the Clock algorithm is implemented by maintaining a referenced bit in each page table entry and a clockHand that points to the current page being considered for eviction. The algorithm iterates over the pages in a circular manner, clearing the referenced bit of each page it skips over until it finds a page to evict.

**Code Snippet:**

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

These two page replacement algorithms, LRU and Clock, are essential components of the virtual memory management system. They help efficiently manage the limited physical memory by determining which pages should be replaced when new pages need to be loaded, ensuring that the system performs well even under constrained memory conditions.

**Memory Management Design**

The memory management design in this project focuses on efficiently handling memory accesses in a simulated virtual memory system. The system ensures that the limited physical memory is effectively utilized by managing how data is read from and written to virtual memory. This is accomplished through a combination of page tables, physical memory management, and page replacement algorithms.

**Overall Memory Management Design:**

* The system uses an integer array to represent physical memory, which is shared by two threads simulating separate processes. Each thread has its own virtual memory space, but both share the same physical memory.
* Virtual memory is larger than physical memory, so excess data is stored on a disk file. When a page is needed that is not currently in physical memory, the system retrieves it from the disk, potentially replacing an existing page in physical memory.

**Memory Access Handling:** Memory accesses are handled using two primary functions: get and set. These functions manage how data is read from and written to virtual memory, triggering the page replacement algorithms when necessary.

**Merge Sort and Memory Access:**

* The merge sort algorithm is used to sort the array of integers stored in virtual memory. This algorithm divides the array into smaller sub-arrays, sorts them, and then merges them back together.
* During the sorting process, the get function is used to retrieve elements from the virtual memory, and the set function is used to store the sorted elements back into the virtual memory.

**How get and set are Used:**

* **get Function in Merge Sort:**
  + The get function retrieves elements from the virtual memory, enabling the merge sort algorithm to compare and sort elements.
  + When the merge sort algorithm needs to access an element in the array, it calls get, which checks if the element is in physical memory. If not, it triggers a page fault to load the required page from disk.
* **set Function in Merge Sort:**
  + The set function is used to write sorted elements back into their correct positions in the virtual memory.
  + After sorting a sub-array, the set function updates the virtual memory with the newly sorted elements, managing any necessary page replacements if the required page is not in physical memory.

**Screenshots and Code Examples**

In this section, we'll include screenshots of the output. Some example of different inputs:

1.input:

./main.o 4 5 7 CL 100 diskFileName.dat

Output:  
metin, ekran görüntüsü, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu  
  
2. ./main.o 4 5 7 LRU 100 diskFileName.dat  
metin, ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu  
input:  
./main.o 8 5 7 LRU 100 diskFileName.dat  
output:  
metin, ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

İnput: ./main.o 8 5 7 LRU 1000 diskFileName.dat  
  
output:  
metin, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

*  In the merge function, the get function is used to retrieve elements from the virtual memory, which are then stored in temporary arrays L and R.
*  The elements are compared and merged back into the main array using the set function, which ensures that the sorted data is written back to the correct locations in virtual memory.
*  The mergeSort function recursively divides the array and sorts it, relying on the get and set functions to handle memory access during the sorting process.

**Conclusion**

In this project, I was required to implement a virtual memory management system using two threads. However, due to an unresolved issue during debugging, I was only able to simulate the system using one thread. Despite this, I set up all the necessary structures to support two threads, including virtual memory and page tables for each thread. The unexpected results during my testing led me to proceed with a single-threaded simulation to ensure the system worked correctly. This allowed me to demonstrate the core functionality of the memory management system, even though it was not fully realized in a multi-threaded context.