UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Proiect la Structura Sistemelor de Calcul

Simularea unui sistem de memorie virtuală în limbajul Java

Nume: Maghiar-Cionca Antoniu
Grupa: 30223

Profesor îndrumător: Neagu Mădălin

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Cuprins

1. Introducere……3
2. Studiul Bibliografic……...4
   2.1. Virtual Memory……4
   2.2. File Buffer……..…..5
   2.3. Hash Map……..……..5
3. Analiza……...5
4. Design……7
5. Implementare……8
   5.1 Memoria Virtuală……..8
   5.2 Page……..……8
   5.3 Interfața……..9
   5.4 Gestiunea fișierului disc……..9
6. Concluzii……9
7. Bibliografie……..10

1. Introducere:

Memoria virtuală este un concept esențial în sistemele de operare și în dezvoltarea software modernă, având un rol semnificativ în gestionarea resurselor de memorie ale unui sistem de calcul. Ea permite executarea eficientă a aplicațiilor, indiferent de cantitatea de memorie RAM disponibilă și de dimensiunea programelor. În următorul proiect am implementat o simulare a unui sistem de memorie virtuală în limbajul de programare Java.

Memoria virtuală este o extensie a memoriei RAM (Random Access Memory) a unui calculator și constă în gestionarea automată a transferului datelor între RAM și discurile de stocare. Memoria virtuală împarte memoria fizică în pagini sau cadre mici și le stochează pe discuri. Atunci când o aplicație are nevoie de o pagină din memorie, aceasta este transferată de pe disc în RAM.

Scopul acestui proiect este de a dezvolta un simulator de memorie virtuală utilizând limbajul de programare Java. Simulatorul va reproduce principalele operațiuni efectuate în cazul unei memorii virtuale, inclusiv regăsirea informației, plasarea unei pagini în memoria principală și înlocuirea unei pagini.

Structura proiectului va include o serie de clase și structuri de date pentru a simula gestionarea memoriei virtuală și fizice. O componentă cheie a proiectului va fi tabela de pagini (Page Table), care va mapa paginile virtuale la cele fizice. De asemenea, vom implementa algoritmi de înlocuire a paginilor pentru a decide ce pagini trebuie eliminate din memoria principală atunci când spațiul este limitat.

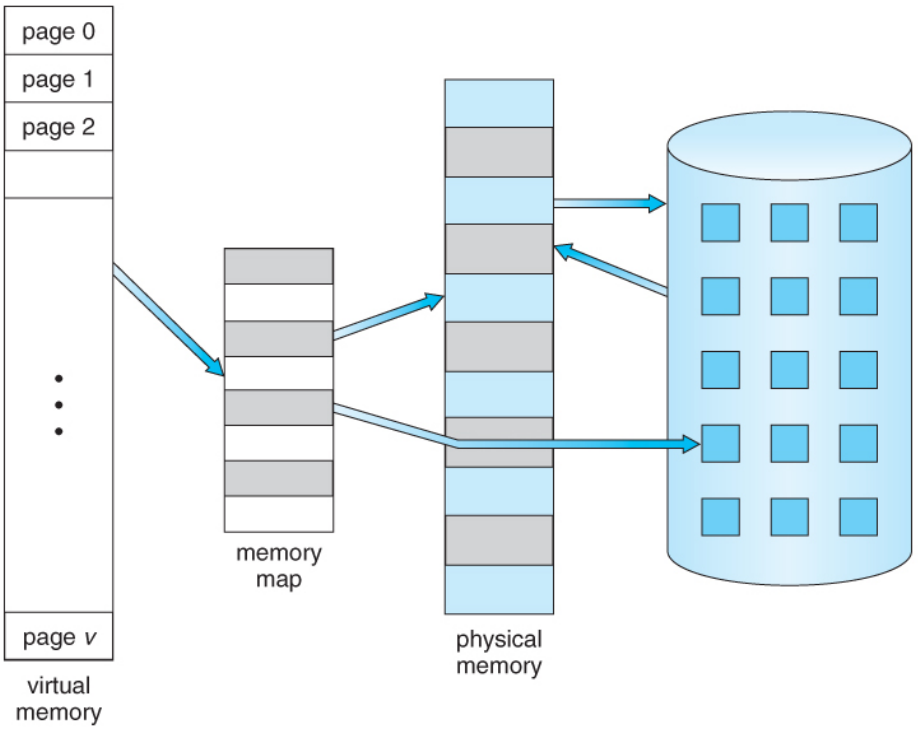
Interacțiunea cu utilizatorul va fi posibilă prin intermediul unei interfețe de utilizator, permițând utilizatorului să efectueze operații specifice de memorie virtuală.

Simulatorul de memorie virtuală dezvoltat în cadrul acestui proiect reprezintă o resursă valoroasă pentru înțelegerea conceptelor de gestionare a memoriei virtuale.

2. Studiul Bibliografic:

2.1. Memoria Virtuala:

Memoria virtuală este un concept în sistemele de operare și arhitectura computerelor care extinde capacitatea de stocare a unui sistem, permițându-i să utilizeze spațiu de stocare pe disc (cum ar fi un hard disk) ca o extensie a memoriei RAM fizice. Acesta este un mecanism care permite aplicațiilor să aibă acces la mai multă memorie decât este fizic disponibilă pe sistem.



2.2 File Buffer:

Un buffer de citire și scriere este o tehnică folosită în programare pentru a îmbunătăți performanța citirii și scrierii de date între două entități, cum ar fi un program și un fișier sau un program și o rețea. Acesta funcționează prin stocarea temporară a datelor într-o zonă de memorie intermediară (buffer) înainte de a fi citite sau scrise efectiv în sursa sau destinația finală. Bufferul servește ca o zonă tampon între cele două entități, reducând astfel numărul de accesări la resursele de stocare și îmbunătățind eficiența operațiunilor.

În Java, clasa BufferedReader este folosită pentru a citi text dintr-un flux de intrare (Reader), oferind avantajul de a utiliza un buffer pentru a reduce numărul de accesări la sursa de date. În mod similar, clasa BufferedWriter în Java permite scrierea eficientă a textului într-un flux de ieșire (Writer), utilizând un buffer.

2.3 Hash Map:

HashMap este o clasă în limbajul de programare Java, inclusă în framework-ul de colecții Java, care implementează interfața Map. Aceasta oferă o structură de date bazată pe perechi cheie-valoare și permite stocarea și recuperarea rapidă a datelor pe baza cheilor. HashMap utilizează o funcție de dispersie (hash function) pentru a converti cheile în indici în cadrul unei tabele de dispersie, ceea ce permite accesul eficient la valorile asociate cheilor.

3. Analiza:

Implementarea memoriei virtuale în Java presupune utilizarea unui fișier pentru reprezentarea memoriei de disc și un HashMap pentru

gestionarea memoriei virtuale, iar aceasta necesită atenție la mai multe aspecte pentru a asigura funcționarea corectă și eficientă a sistemului.

În cadrul HashMap-ului, paginile sunt reprezentate și gestionate, iar operațiile de adăugare, actualizare și ștergere sunt implementate pentru a menține coerența datelor în memoria virtuală.

Datele sunt stocate pe disc într-un fișier, iar operațiile de citire și scriere sunt implementate pentru a sincroniza starea actuală a memoriei virtuale cu cea a memoriei de disc.

Interfața grafică a aplicației permite utilizatorului să interacționeze cu memoria virtuală, iar conținutul HashMap-ului este afișat în mod corespunzător.

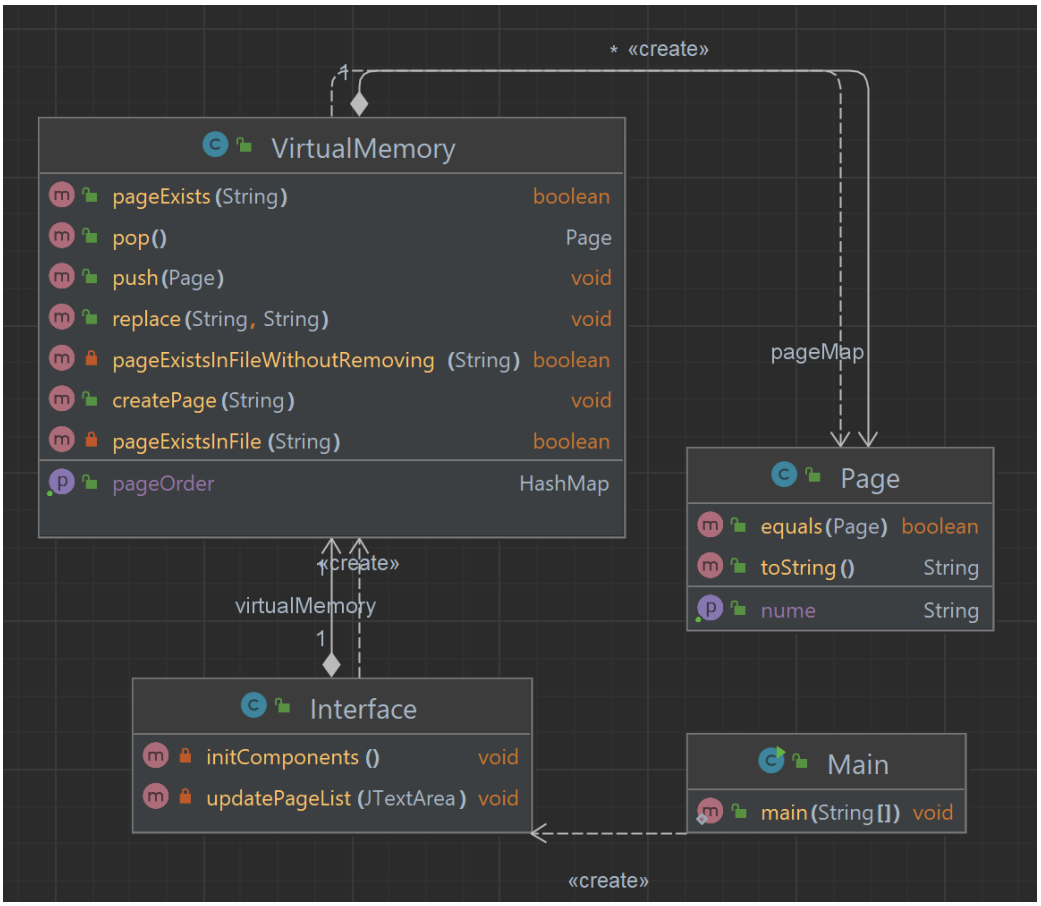
Operarea cu fereastra de fișier este gestionată pentru a asigura persistența datelor pe disc, evitând pierderea acestora în situații neașteptate, precum închiderea bruscă a aplicației.

Optimizarea performanței este abordată prin implementarea unor strategii eficiente pentru accesul la date în HashMap și prin minimizarea costului operațiilor de citire și scriere în fișier pentru a evita impactul asupra performanței.

Tratarea excepțiilor și a situațiilor anormale este integrată în logica aplicației, asigurând astfel integritatea datelor și funcționarea robustă în scenarii neașteptate.

4. Design:

Diagrama UML pe baza căruia a fost implementa simularea este următoarea:



5. Implementare:

5.1 Memoria Virtuala:

Atribute:

- PageMap: un HashMap ce reprezintă memoria virtuala cu cheile fiind numele paginilor și valorile fiind obiecte de tipul Page
- PageOrder: o listă ce păstrează ordinea paginilor în memoria virtuală

Constructor:

Constructorul inițializează PageMap-ul cu o capacitate maximă de 10 elemente, reflectând dimensiunea maximă a memoriei virtuale. PageOrder este inițializat ca o lista goală.

Metodele principale:

- Push: adaugă o pagină în memoria virtuală
- Pop: elimină cea mai veche pagină din memorie și o adauga la sfârșitul fișierului disc
- createPage: crează o nouă pagina în memoria virtuală dacă aceasta există pe disc și memoria virtuală nu este plină
- Replace: înlocuiește o pagina din memoria virtuală cu una de pe disc

5.2 Page:

Clasa Page a fost definită pentru a reprezenta paginile din memoria virtuală.

5.3 Interfața:

Interfața grafică conferă interacțiunea utilizatorului cu memoria virtuală și operații de tipul adăugare, ștergere și înlocuire asupra paginilor memoriei virtuale cât și celor de pe disc.

Starea curentă a memoriei virtuale este afișată în timp real în cadrul interfeței grafice.

5.4 Gestiunea fișierului disc:

Fișierul de disc “DiscMem.txt” este utilizat pentru a simula datele de pe disc.

6. Concluzii:

Implementarea memoriei virtuale în limbajul de programare Java furnizează o soluție robustă și eficientă pentru gestionarea resurselor de memorie în cadrul unei aplicații interactive. Prin utilizarea unui HashMap pentru reprezentarea memoriei virtuale și a unui fișier pentru persistența datelor, această implementare îmbină beneficiile unei structuri de date eficiente cu necesitatea păstrării coerenței datelor între sesiunile aplicației.

7. Bibliografie:

- https://www.techtarget.com/searchstorage/definition/virtual-memory
- https://www.geeksforgeeks.org/virtual-memory-in-operating-system/
- https://codegym.cc/groups/posts/bufferedreader-and-bufferedwriter
- https://www.javatpoint.com/java-swing