Proiect Structura Sistemelor de Calcul

-Simulator de memorie virtuala-

Moldovan Claudia

Grupa 30234

An universitar 2022-2023

Profesor coordonator: Neagu Madalin

Cuprins

1.Cerinta proiectului

Se va scrie un program de simulare grafica a principalelor operatii efectuate in cazul unei memorii virtuale: regasirea informatiei, plasarea unei pagini in memoria principala, inlocuirea unei pagini (Java, C, C++, C# etc.)

2.Introducere

Vom începe turul subiectului, definind programul de simulare**,** care nu este altceva decât faptul de a emula comportamentul sau funcționarea unui sistem original într-o perioadă de timp; care se poate face manual sau computerizat.

Cu alte cuvinte, este un model sau un set de ipoteze care ne permit să stabilim o comparație a comportamentului real față de ceea ce putem observa într-un timp dat. Aceste tipuri de ipoteze trebuie exprimate în ecuații logice și matematice între entități sau entități.

În astfel de scopuri, pentru a realiza o simulare în domeniul tehnologic, este necesar să se ia în considerare tipurile de limbaje care sunt utilizate în acest scop, care au un scop sau scop specific, care este să permită capacitatea echipamentelor de calcul să fie crescut la un cost redus, însă simularea nu este întotdeauna recomandabilă. De aceea, următoarele precizează de ce și pentru ce trebuie să simulăm:

- Permite efectuarea unui studiu bazat pe expertiza și interacțiunea cu sistemele interne ale computerului.
- Permite prin observație cunoașterea modificărilor care apar în comportamentul unui sistem.
- Devine mult mai usor sa proiectezi un model de simulare care sa acopere nevoile din punctul de vedere al cunostintelor pe care le ai cu privire la sistemul de studiu.
- Acesta servește ca instrument pedagogic deoarece permite utilizatorului să consolideze posibilele soluții teoretice care sunt disponibile pe sistemul de studiu.
- Determinați capacitățile hardware ale echipamentului pentru a îndeplini cerințele.

Memoria virtuala este un mediu de stocare temporar folosit de un computer pentru a rula programe care necesita mai multa memorie decat are sistemul.

Atunci cand computerul ramane fara memorie cu acces aleator (RAM) necesara pentru a rula un program sau o operatie , Windows foloseste memoria virtuala pentru a compensa.

Memoria virtuala combina RAM-ul sistemului cu spatiul de pe hard. Cand RAM-ul devine insuficient, memoria virtuala muta informatii/date din RAM catre un spatiu, numit page file. Cu alte cuvinte, mutand informatii catre si din page file elibereaza destul RAM pentru ca programele sa poata termina operatiile.

- Formula de calcul a valorii minime pentru sistemele cu cel mult 1GB de memorie RAM e RAM*1.5. Astfel, dacă sistemul dispune de 512 MB de memorie RAM, dimensiunea minimă a memoriei virtuale trebuie să fie 768 MB (512*1.5).
- Formula de calcul a valorii minime pentru sisteme cu mai mult de 1GB de memorie RAM este RAM+300 MB. Astfel, dacă sistemul dispune de 2GB de memorie RAM, valoarea minimă va fi 2340 MB (2048+300).
- Formula de calcul a valorii maxime este RAM*3. Astfel, pentru un sistem cu 512 MB de memorie RAM instalată, valoarea maximă va fi 1536 MB (512*3).

3.Studiu bibliografic

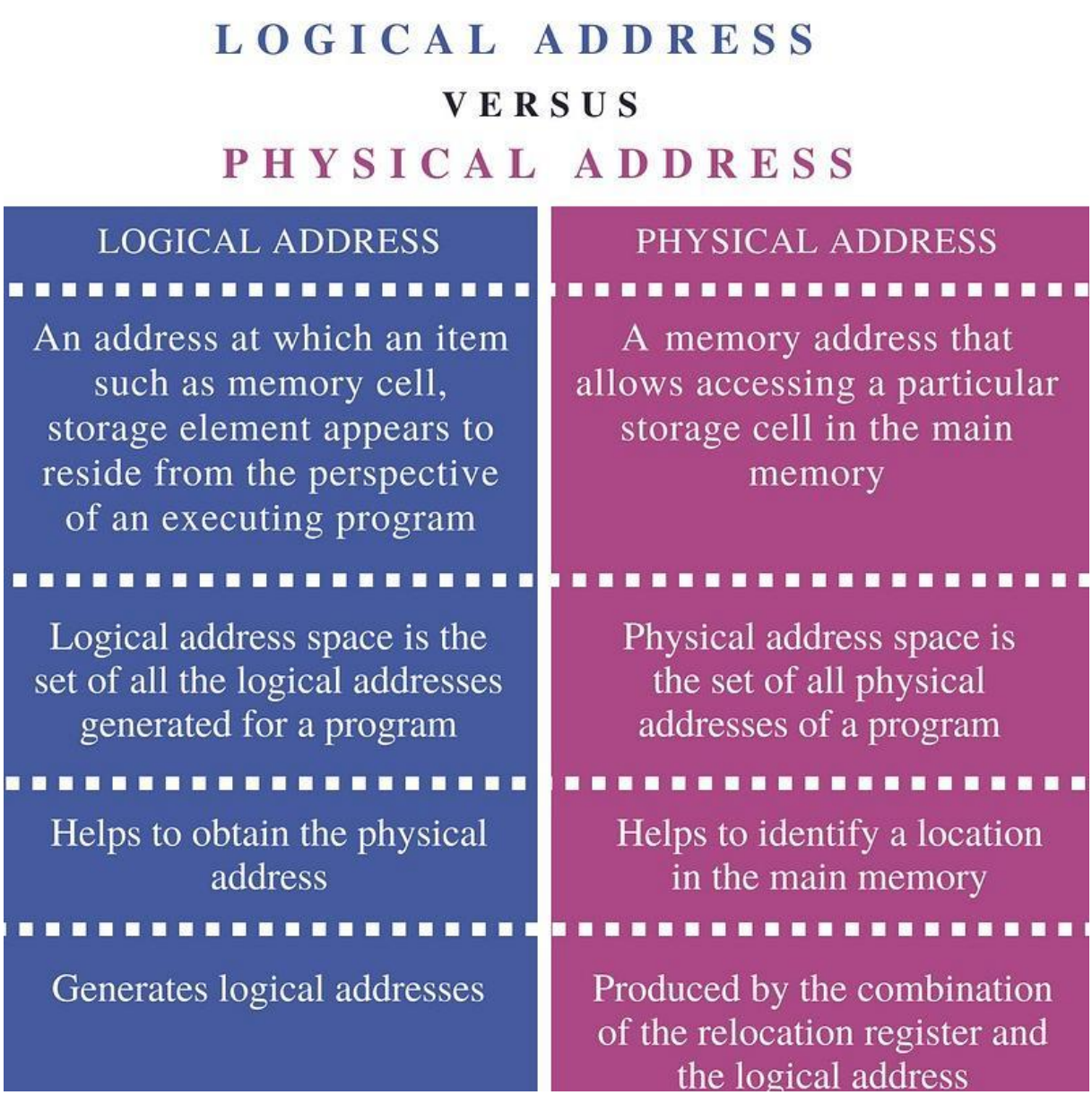
Pentru a realiza acest proiect primii pasi sunt studierea si aflarea partii teoretice, pe care mai departe vom dori sa o implementam.

Eu m-am gandit sa urmaresc urmatoarele 4 operatii in lucrul cu memoria virtuala: transformarea adresei logice intr-o adresa fizica, utilizarea memoriei fizice, a page tabel-ului si a TLB-ului.

Luandu-le pe fiecare in parte pentru a le studia am aflat urmatoarele pentru fiecare cerinta:

I. Transformarea adresei virtuale in adresa fizica

Există două tipuri de adrese cunoscute ca adrese logice și adrese fizice. CPU generează adresa logică. Este, de asemenea, denumită o adresă virtuală. Adresa fizică este o adresă reală în unitatea de memorie. Accesarea unui program din memoria secundară consumă mai mult timp. Prin urmare, programul se încarcă în memoria principală în momentul execuției. Apoi, adresa logică generată de CPU se convertește într-o adresă fizică pentru a găsi locația corespunzătoare din memoria principală.



II. Memorie fizica

Memoria fizica se refera la memoria RAM și hard disk-urile din computerul utilizat pentru stocarea datelor. Pe un computer, sistemul de operare, programele de aplicație și datele utilizate în prezent sunt păstrate în memoria RAM, astfel încât acestea să poată fi accesate rapid de către procesor. RAM ar putea fi accesat mai repede decât celelalte dispozitive de stocare, cum ar fi hard disk-ul și CD-ROM-ul. Dar datele din memoria RAM există numai în timp ce computerul este în funcțiune. Când alimentarea este oprită, toate datele din memoria RAM sunt pierdute, iar

sistemul de operare și alte date sunt încărcate din nou în memoria RAM de pe hard disk când calculatorul este pornit. Hard diskul este o memorie nevolatilă (o memorie care păstrează datele chiar și atunci când nu este alimentată) care este utilizată pentru a stoca date pe un computer. Este alcătuită din discuri circulare numite platouri care stochează date magnetice. Datele sunt scrise și citite în și de la platouri folosind capete de citire / scriere.

III. Page tabel

Un page tabel este structura de date utilizată de un sistem de memorie virtuală într-un sistem de operare pentru a stoca maparea între adrese virtuale și adrese fizice. Adresele virtuale sunt utilizate de programul executat prin procesul de accesare, în timp ce adresele fizice sunt utilizate de hardware sau, mai precis, de subsistemul de memorie cu acces aleatoriu (RAM). Page tabel este o componentă cheie a traducerii virtuale a adreselor, care este necesară pentru a accesa datele din memorie.

IV. TLB

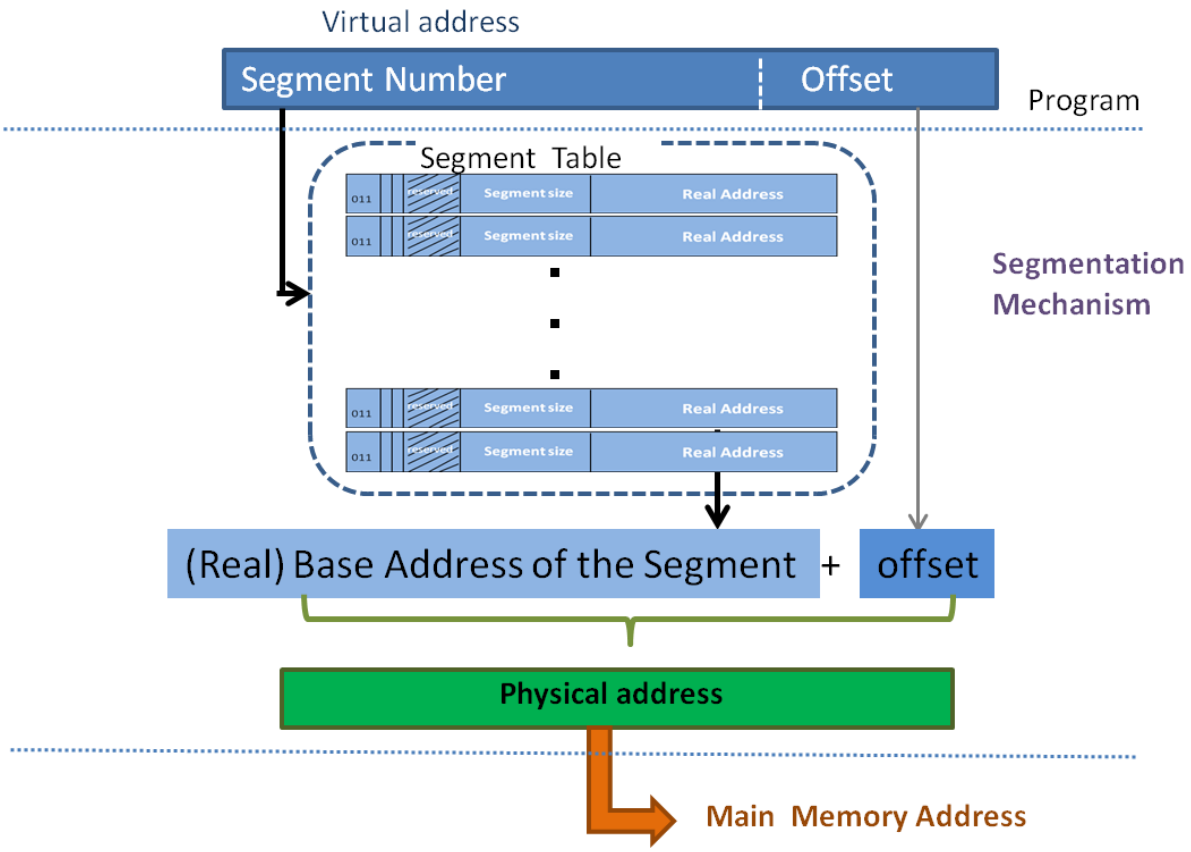
Un translation lookaside buffer (TLB) este un cache de memorie care stochează translatiile recente ale memoriei virtuale în memoria fizică. Acesta este folosit pentru a reduce timpul necesar pentru a accesa o locație de memorie a utilizatorului. Poate fi numit un cache de traducere a adresei. Face parte din unitatea de gestionare a memoriei cipului (MMMU) . Un TLB poate fi situat între CPU și memoria cache a procesorului, între memoria cache a procesorului și memoria principală sau între diferitele niveluri ale memoriei cache pe mai multe niveluri. Majoritatea procesoarelor desktop, laptop și server includ unul sau mai multe TLB-uri în hardware-ul de gestionare a memoriei și este aproape întotdeauna prezent în orice procesor care utilizează memorie virtuală segmentată sau în paged.

4.Design

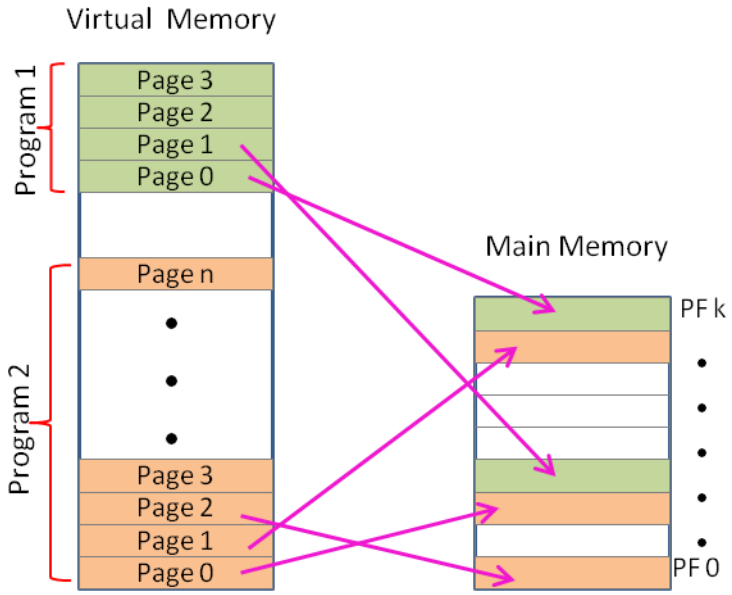
Ca si idee proiectul v-a avea urmatorul design. La rulare se v-a deschide fereastra de meniu din care vom putea alege intre operatiunile pe care dorim sa le facem: translate adress, page tabel, tbl, memorie fizica, plasare in memorie, plasarea paginii in memorie, replasarea paginii in memorie.

La translatarea adresei, odata ce vom sti adresa virtuala, avand dimensiunea paginii de 4KB, vom calcula page number cu formula : page_number=virtual_adress/page_size si offsetul cu formula offset=virtual_adress % page_size. La final rezultatul, adica adresa fizica v-a fi determinata de adresa_fizica (memory.size()) + offset.

Din moment ce nu cunoaștem adresa fizică în prealabil, nu putem fi pe deplin siguri de acuratețea translatiei.



Paging este o altă implementare a memoriei virtuale. Stocarea logică este marcată ca pagini de o anumită dimensiune, să zicem 4KB. MM(Main Memory) este vizualizat și numerotat ca si cadre de pagină. Fiecare cadru de pagină este egal cu dimensiunea paginilor. Paginile din vizualizarea logică sunt montate în cadrele de pagină goale din MM Acest lucru este sinonim cu plasarea unei cărți într-un raft. De asemenea, conceptul este similar cu blocurile cache și plasarea lor.

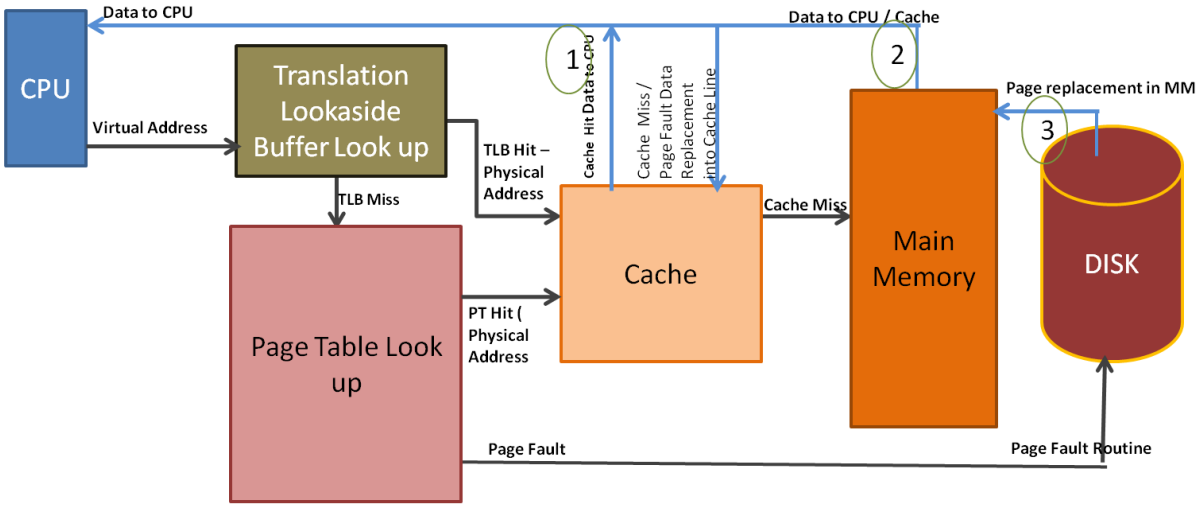


Această mapare este necesară pentru a fi menținută într-un tabel de pagini. Maparea este utilizată în timpul traducerii adresei. De obicei, un tabel de pagini conține adresa virtuală a paginii, numărul de cadru fizic corespunzător în care este stocată pagina, bitul de prezență, bitul de schimbare și drepturile de acces. Acest tabel de pagini este referit pentru a verifica dacă pagina dorită este disponibilă în MM .Tabelul de pagini se află într-o parte a MM Astfel, fiecare acces la memorie solicitat de CPU va trimite memoria de două ori – o dată la tabelul de pagini și a doua oară pentru a obține datele din locația accesată.

Pentru implementarea page-tabel-ului voi lua in considerare si alocarea/deaclocarea unei pagini.

Intrările TLB sunt similare cu cele din tabelul de pagini. Odată cu includerea TLB, fiecare adresă virtuală este verificată inițial în TLB pentru traducerea adreselor. Dacă este un TLB Miss, atunci tabelul de pagini în MM este analizat. Astfel, un TLB Miss nu provoacă eroare de pagină. Eroarea de pagină va fi generată numai dacă este o eroare și în tabelul de pagini, dar nu în alt mod. Deoarece TLB este un cache de adresă asociativă

în CPU, TLB Hit oferă cea mai rapidă traducere posibilă a adresei; cel mai bun următorul este pagina hit în Page Table; cel mai rău este eroarea de pagină.



Odată ce adresa este tradusă într-o adresă fizică, atunci datele sunt deservite la CPU. Există trei posibilități, în funcție de locul în care se află datele.

Cazul 1 - TLB sau PT hit și, de asemenea, Cache hit - date returnate de la CPU la Cache

Cazul 2 - TLB sau PT hit și Cache Miss - datele returnate de LA MM la CPU și Cache

Caz 3 - eroare pagină - date de pe disc încărcate într-un segment / Cadrul paginii în MM; MM returnează datele în CPU și cache

La plasarea unei pagini trebuie sa se creeze page-tabel-ul care sa poata contine numarul paginii vrituale si numarul paginii fizice. Astfel la implementare sa se poata plasa o pagina in memorie in functie de cele doua numere.

La replasarea pagini, prima data vom initializa memoria principala cu pagini din page tabel , iar apoi vom replasa pagina dorita.

Din fiecare operatiune de meniu, se va putea da back pentru a alege o noua cerinta.

5.Implementation

A. Page Table

La implementare am folosit o structura PageTableEntry pentru detaliile paginii si o clasa PageTabel in care am introdus toate functiile necesare rezolvarii acestui punct. Am implementat urmatoarele functii: findPage care gaseste o pagina in pageTable, allocatePage care aloca o noua pagina in pageTable, deallocatePage ce dealoca/sterge pagina din PageTable, o functie ce afiseaza PageTable si o functie generala in care apelez functiile de mai sus.

```
struct PageTableEntry {
    int pageNumber;
    bool valid;
    int frameNumber;
};

class PageTable {
private:
    std::vector<PageTableEntry> entries;
    int numFrames;

public:
    PageTable(int numFrames) : numFrames(numFrames) {
        for (int i = 0; i < numFrames; i++) {
            PageTableEntry entry = { -1, false, -1 };
            entries.push_back(entry);
        }
    }

    PageTableEntry* findPage(int pageNumber) {
        for (int i = 0; i < numFrames; i++) {
            if (entries[i].pageNumber == pageNumber && entries[i].valid) {
                return &entries[i];
            }
        }
        return nullptr;
    }


    bool allocatePage(int pageNumber, int frameNumber) {
        PageTableEntry* entry = findPage(pageNumber);
        if (entry) {
            return false;
        }
        for (int i = 0; i < numFrames; i++) {
            if (!entries[i].valid) {
                entries[i] = { pageNumber, true, frameNumber };
                return true;
            }
        }
        return false;
    }
```

```
    bool deallocatePage(int pageNumber) {
        PageTableEntry* entry = findPage(pageNumber);
        if (entry) {
            entry->valid = false;
            return true;
        }
        return false;
    }
    void printPageTable() {
        std::cout << "Page Number\tValid\tFrame Number" << std::endl;
        for (int i = 0; i < numFrames; i++) {
            std::cout << entries[i].pageNumber << "\t\t" << entries[i].valid
                << "\t\t" << entries[i].frameNumber << std::endl;
        }
    }
};
void PageTableFunction()
{
    PageTable pt(3);
    pt.allocatePage(1, 0);
    pt.allocatePage(2, 1);
    pt.allocatePage(3, 2);
    pt.printPageTable();
    pt.deallocatePage(2);
    pt.printPageTable();
       }
```

B. Translatarea adresei din virtuala in fizica

Pentru implementare avem nevoie de pageTable initializat sub forma de map pentru maparea adresei virtuale si fizice si o memorie sub vorba unui vector de bytes. Functia de translatare a adresei face acest lucru posibil prin utilizarea pageTabel-ului. Daca adresa nu este prezenta in memorie se face page faul si se aduce pagina in memorie. Adresa fizica se calculeaza in functie de offset, acesta din urma fiind calculat pe vaza adresei virtuale si a dimensiunii paginii.

```
std::unordered_map<size_t, size_t> page_table;
std::vector<uint8_t> memory;
 size_t translate_address(size_t virtual_address)
{
    auto it = page_table.find(virtual_address);
    if (it == page_table.end()) {
        size_t page_size = 4096;
        size_t page_number = virtual_address / page_size;
        size_t offset = virtual_address % page_size;
        size_t physical_address = memory.size();
        memory.resize(physical_address + page_size);
        page_table[virtual_address] = physical_address;
        return physical_address + offset;
    }
    return it->second;
}
 void Translate()
 {
```

```
    std::cout << std::hex << translate_address(0x1000) << std::endl; // prints 0
    std::cout << std::hex << translate_address(0x2000) << std::endl; // prints 1000
    std::cout << std::hex << translate_address(0x3000) << std::endl; // prints 2000
    std::cout << std::hex << translate_address(0x4000) << std::endl; // prints 3000
       }
```

C. TLB

Se fac declaratiile de inceput a pageTabel-ului si a memoriei asemanatoare cu situatia de la punctul B, in plus se adauga tlb tot sub forma de map. O functie calculeaza adresa fizica in functie de tlb si de pageTabel. Diferenta dintre cele doua abordari este ca in acest caz se cauta adresa virtuala in Tlb, iar daca aceasta nu este prezenta acolo, se cauta in page tabel. Daca este prezenta in page-tabel este adauagata in tlb si este calculata adresa fizica.

```
std::unordered_map<size_t, size_t> page_table_tlb;
std::vector<uint8_t> memory_tlb;
std::unordered_map<size_t, size_t> tlb;
size_t translate_address_tlb(size_t virtual_address_tlb)
{
    auto it = tlb.find(virtual_address_tlb);
    if (it != tlb.end()) {
        return it->second;
    }
    it = page_table_tlb.find(virtual_address_tlb);
    if (it == page_table_tlb.end()) {
        size_t page_size = 4096; // 4KB pages
        size_t page_number = virtual_address_tlb / page_size;
        size_t offset = virtual_address_tlb % page_size;
        size_t physical_address = memory_tlb.size();
        memory_tlb.resize(physical_address + page_size);
        page_table_tlb[virtual_address_tlb] = physical_address;
        return physical_address + offset;
    }
    tlb[virtual_address_tlb] = it->second;
    return it->second;
}

void Tlb()
{

    std::cout << std::hex << translate_address_tlb(0x1000) << std::endl;
    std::cout << std::hex << translate_address_tlb(0x2000) << std::endl;
    std::cout << std::hex << translate_address_tlb(0x3000) << std::endl;
    std::cout << std::hex << translate_address_tlb(0x4000) << std::endl;
}
```

D. Memoria fizica

In implementare am abordat o solutie usoara, in care voi adauga simplu valori in memorie.

```
const int PHYSICAL_MEMORY_SIZE = 1024;
void PhysicalMemory()
{
    int physical_memory[PHYSICAL_MEMORY_SIZE];
    for (int i = 0; i < PHYSICAL_MEMORY_SIZE; i++) {
        physical_memory[i] = i;
    }
    for (int i = 0; i < PHYSICAL_MEMORY_SIZE; i++) {
        std::cout << "Physical memory location " << i << ": " << physical_memory[i] <<
std::endl;
    }
       }
```

E. Plasarea in memorie

Declar in mod obisnuit ca pana acum memoria de tip map, plasez valori in ea si preiau o valoare de la o anumita adresa.

```
void placing()
{
    std::map<int, int> virtual_memory;
    for (int i = 0; i < 1024; i++) {
        virtual_memory[i] = i;
    }
    int virtual_address = 42;
    int value = virtual_memory[virtual_address];
    std::cout << "Value at virtual memory address " << virtual_address << ": " <<
value << std::endl;

       }
```

F. Plasarea unei pagini in memorie

Pentru plasarea paginii in memorie am avut nevoie de dimensiunea unei pagini si numarul de pagini din memoria virtuala. Astfel am creat un map ce sa reprezinte pageTabel-ul pe care l-am initializat cu fiecare numar de pagina. Dupa care am plasat o pagina anume in memoria principala.

```
const int PAGE_SIZE = 4096;
const int NUM_PAGES = 100;
void placing_page()
{
    std::map<int, int> page_table;
    for (int i = 0; i < NUM_PAGES; i++) {
        page_table[i] = i;
    }
    int virtual_page_number = 42;
```

```
    int physical_page_number = 10;
    page_table[virtual_page_number] = physical_page_number;
    for (int i = 0; i < NUM_PAGES; i++) {
        std::cout << "Virtual page " << i << " is mapped to physical page " <<
page_table[i] << std::endl;
    }

       }
```

G. Replasarea unei pagini

Plasez cateva pagini in pageTable, pe care mai apoi le iau in memoria principala, Dupa care dau pagina pe care vreau sa o inlocuiesc si cea cu care o inlocuiesc o sterg si dau push_back la noua pagina.

```
void replacing()
{
    std::map<int, int> page_table;
    for (int i = 0; i < NUM_PAGES; i++) {
        page_table[i] = -1;
    }
    page_table[0] = 0;
    page_table[1] = 1;
    page_table[2] = 2;
    page_table[3] = 3;
    std::list<int> main_memory;
    for (int i = 0; i < NUM_PAGES; i++) {
        if (page_table[i] != -1) {
            main_memory.push_back(page_table[i]);
        }
    }
    std::cout << "First Main memory: ";
    for (int page : main_memory) {
        std::cout << page << " ";
    }
    std::cout << std::endl;
    int virtual_page_number = 2;
    int physical_page_number = 10;
    page_table[virtual_page_number] = physical_page_number;
    main_memory.remove(virtual_page_number);
    main_memory.push_back(physical_page_number);
    std::cout << "After replace Main memory: ";
    for (int page : main_memory) {
        std::cout << page << " ";
    }
    std::cout << std::endl;
       }
```

H. Am ales pentru o interfata cat de cat prietenoasa sa implementez si un Meniu, despre care nu voi explica, deoarece consider ca nu tine de materie.

6.Teste

```
C:\Users\Claud\Desktop\ProjectSSC\Debug\ProjectSSC.exe
Menu
Press 1 for Page Table
Press 2 for Translate from virtual adress to physical adress
Press 3 for Translate with Tlb
Press 4 for Physical Memory
Press 5 for Placing in memory
Press 6 for Placing a page in main memory
Press 7 for Replacing page
```

PRESS 1

```
1
Page Number     Valid   Frame Number
1               1               0
2               1               1
3               1               2
Page Number     Valid   Frame Number
1               1               0
2               0               1
3               1               2

------------------------------
For back press 0
```

Am alocat paginile 1,2,3 cu frame 0,1,2 ceea ce inseamna ca sunt valide toate

Am dealocat pagina 2, ceea ce a facut ca aceasta sa devina invalida

PRESS 0 -> PRESS 2

```
2
0
1000
2000
3000
```

In urma formulelor aplicate, pentru adresele virtuale 0x1000,0x2000,0x3000,0x4000 se obtin rezultatele 0,1000,2000,3000.

PRESS 0 -> PRESS 3

```
3
0
1000
2000
3000

------------------------------
For back press 0
```

Folosind aceleasi adrese virtuale ca cele de mai sus, insa metoda diferita se obtin aceleasi rezultate

PRESS 0 -> PRESS 4

```
4
Physical memory location 0: 0
Physical memory location 1: 1
Physical memory location 2: 2
Physical memory location 3: 3
Physical memory location 4: 4
Physical memory location 5: 5
Physical memory location 6: 6
Physical memory location 7: 7
Physical memory location 8: 8
Physical memory location 9: 9
Physical memory location a: a
Physical memory location b: b
Physical memory location c: c
Physical memory location d: d
Physical memory location e: e
Physical memory location f: f
Physical memory location 10: 10
Physical memory location 11: 11
```

Am ales sa adaug in locatiile din memorie valori exact de la 0 pana la dimensiune memoriei

PRESS 0 -> PRESS 6

```
6
Virtual page 0 is mapped to physical page 0
Virtual page 1 is mapped to physical page 1
Virtual page 2 is mapped to physical page 2
Virtual page 3 is mapped to physical page 3
Virtual page 4 is mapped to physical page 4
Virtual page 5 is mapped to physical page 5
Virtual page 6 is mapped to physical page 6
Virtual page 7 is mapped to physical page 7
Virtual page 8 is mapped to physical page 8
Virtual page 9 is mapped to physical page 9
Virtual page a is mapped to physical page a
Virtual page b is mapped to physical page b
Virtual page c is mapped to physical page c
Virtual page d is mapped to physical page d
Virtual page e is mapped to physical page e
Virtual page f is mapped to physical page f
Virtual page 10 is mapped to physical page 10
Virtual page 11 is mapped to physical page 11
Virtual page 12 is mapped to physical page 12
Virtual page 13 is mapped to physical page 13
```

PRESS 0 -> PRESS 7

```
7
First Main memory: 0 1 2 3
After replace Main memory: 0 1 3 10

------------------------------
For back press 0

```

Am introdus paginile 0,1,2,3 in memorie si am ales sa replasez in locul paginii 2 pagina 10.