

Tema 3 Memorie virtuală

Data publicare: **6 Aprilie 2017**



Deadline: **26 Aprilie 2017, ora 23:55**

Deadline hard: **3 Mai 2017, ora 23:55**

Obiectivele temei

- Aprofundarea conceptelor legate de mecanismele memoriei virtuale.
- Obținerea de deprinderi pentru lucrul cu excepții de memorie pe sistemele Linux și Windows.
- Aprofundarea API-ului Linux și Windows de lucru cu spațiul de adrese și memoria virtuală.





Recomandări

- Înainte de a începe implementarea temei este recomandată acomodarea cu noțiunile și conceptele specifice, precum:
 - memorie virtuală
 - memorie fizică (RAM)
 - spațiu de adresă
 - pagini virtuale – *pages*
 - pagini fizice – *frames*
 - tabelă de pagini
 - maparea unei pagini virtuale peste o pagină fizică
 - *page fault*
 - drepturi de acces la pagină
 - *demand paging*
 - spațiu de swap
 - *swap in*
 - *swap out* (evacuare)
 - înlocuirea unei pagini – *page replacement*
 - maparea fișierelor – *file mapping*
- Urmăriți resursele descrise în secțiunea [Resurse de suport](#).

Enunț

Să se implementeze sub formă unei biblioteci partajate/dinamice un **swapper & demand pager**. Biblioteca va permite alocarea de multiple zone de memorie virtuală pentru care va simula operații specifice memoriei virtuale: *page fault*, *demand paging*, *swap in*, *swap out*.

Pentru fiecare zonă alocată, diversele componente se vor simula în felul următor:

- *Memoria virtuală* va fi reprezentată de o zonă de memorie din spațiul de adresă al procesului.
 - Veți folosi funcțiile din familia  *mmap* (Linux) și  *VirtualAlloc* (Windows).
- *Memoria fizică* (memoria RAM) va fi simulată de un fișier. Simularea mapării unei pagini de memorie virtuală (alocarea unei pagini fizice aferente) se va realiza prin maparea acesteia peste o zonă asociată din fișierul RAM.
 - Veți folosi funcțiile  *mmap* (Linux) și  *MapViewOfFileEx* (Windows) pentru maparea paginilor virtuale peste "pagina" aferentă din cadrul

Informații generale SO

- Documentație și alte resurse
- Feed RSS
- Hall of SO
- Listă de discuții
- Mașini virtuale
- Trimitere teme

Informații SO 2016-2017

▼ Examen

- Examen CA/CC 2012-2013
- Examen CA/CC 2013-2014
- Examen CA/CC 2014-2015
- Examen CA/CC 2015-2016

▼ Reguli generale și notare

- Notare CA/CB/CC
- Anunțuri
- Calendar
- Catalog
- Echivalări teme
- Karma Awards
- SO Need to Know
- Orar și împărțire pe semigrupe

Laboratoare

▼ Resurse

- C/SO Tips
- Macro-ul DIE
- GDB
- Resurse
- Function Hooking and Windows DLL Injection
- Oprofile
- Recapitulare
- Thread-uri - Extra
- Visual Studio Tips and Tricks
- windows-video
- Laborator 01 - Introducere
- Laborator 02 - Operații I/O simple
- Laborator 03 - Procese
- Laborator 04 - Semnale
- Laborator 05 - Gestiunea memoriei

fișierului RAM.

- *Spațiul de swap* va fi simulat de un alt fișier cu dimensiunea egală cu a spațiului virtual de adresă. Fiecărei pagini virtuale îi va corespunde o "pagină" în cadrul fișierului ce simulează spațiul de swap.

Exprimarea "pagină în cadrul unui fișier" se referă la o regiune de dimensiunea unei pagini din cadrul fișierului. Fișierul RAM are dimensiunea a `num_frames` pagini (`num_frames` este numărul de pagini fizice) iar fișierul aferent spațiului de swap are dimensiunea a `num_pages` pagini (`num_pages` este numărul de pagini virtuale).

Simulatorul va oferi următoarele funcționalități:

- Alocarea de multiple zone de memorie virtuală; fiecare zonă va fi asociată cu un fișier care simulează memoria RAM și un fișier care simulează spațiul de swap.
- Pentru fiecare zonă, spațiul de swap are dimensiunea spațiului de memorie virtuală. Dimensiunea spațiului de memorie virtuală (numărul de pagini virtuale – *pages*) trebuie să fie mai mare sau egală cu dimensiunea memoriei RAM (numărul de pagini fizice – *frames*).
- La un acces la o pagină de memorie virtuală alocată dar nemapată peste RAM, se generează un *page fault*, corespunzător procesului de *demand paging*. În urma *page fault*-ului, pagina de memorie virtuală este mapată peste o pagină din RAM (peste fișierul aferent).
- În momentul în care o pagină virtuală este mapată peste o pagină fizică în urma procesului de *demand paging* pagina fizică trebuie "curățată" (adică "umplută" cu valori de zero).
- Paginile virtuale se consideră inițial fără protecție. În momentul generării unui *page fault* și a alocării unei pagini fizice, pagina virtuală este marcată *read-only*. Dacă accesul a fost de scriere, se va genera un nou *page fault* și se va marca pagina *read-write*. Astfel, un acces de citire la o pagină nemapată va genera un *page fault*, iar pagina va fi mapată și marcată *read-only*. Un acces de scriere la o pagină nemapată va genera *două page fault-uri*, iar pagina va fi mapată și marcată *read-write*. Un acces de scriere la o pagină mapată și marcată *read-only* va genera un *page fault*, iar pagina va fi marcată *read-write*. O pagină marcată *read-write* este marcată și *dirty* (a fost modificată).
- Dacă în urma unui *page fault* este nevoie de o pagină fizică iar memoria RAM este ocupată, o pagină fizică trebuie evacuată (*swap out*).
 - Nu se impune nici o constrângere asupra algoritmului de înlocuire a paginii (se poate substitui mereu aceeași pagină, o pagină aleatoare, FIFO, LRU etc.).
- Paginile virtuale mapate pot fi *dirty* sau nu. O pagină *dirty* este o pagină care a fost modificată din momentul prezenței acesteia în RAM.
 - Paginile curate (*non-dirty*) selectate pentru a fi înlocuite nu vor fi evacuate (*swap out*). Adică vor fi înlocuite în RAM dar conținutul lor nu va fi copiat în swap, eliminând astfel overhead-ul de copiere.
 - **Excepție:** dacă o pagină este curată (*non-dirty*) dar nu a fost niciodată evacuată (a fost alocată prin *demand paging*), atunci va fi evacuată (*swap out*).
- Dacă o pagină virtuală (**P**) are conținutul în spațiul de swap, la accesarea acesteia se generează *page fault* și pagina trebuie adusă în RAM (*swap in*). În general, *page fault*-ul va genera eliberarea unei pagini fizice din RAM (**F**) (rularea unui algoritm de înlocuire de pagină și *swap out*); peste pagina fizică (**F**) astfel eliberată se va mapa pagina (**P**) la accesul căreia s-a generat *page fault*-ul.

Interfața bibliotecii

Interfața de utilizare a bibliotecii este prezentată în cadrul fișierul header `vmsim.h`. Acesta conține funcții de inițializare și cleanup a bibliotecii (`vmsim_init/vmsim_cleanup`) și funcții de alocare și eliberare a zonelor de memorie (`vm_alloc/vm_free`).



- Laborator 06 - Memoria virtuală
- Laborator 07 - Profiling & Debugging
- Laborator 08 - Thread-uri Linux
- Laborator 09 - Thread-uri Windows
- Laborator 10 - Operații IO avansate - Windows
- Laborator 11 - Operații IO avansate - Linux
- Laborator 12 - Implementarea sistemelor de fișiere

Cursuri

- ▶ Curs 01 - Introducere
- ▶ Curs 02 - Sistemul de fișiere
- ▶ Curs 03 - Procese
- ▶ Curs 04 - Planificarea execuției. IPC
- ▶ Curs 05 - Gestiunea memoriei
- ▶ Curs 06 - Memoria virtuală
- ▶ Curs 07 - Securitatea memoriei
- ▶ Curs 08 - Fire de execuție
- ▶ Curs 09 - Sincronizare
- ▶ Curs 10 - Dispozitive de intrare/ieșire
- ▶ Curs 11 - Networking în sistemul de operare
- ▶ Curs 12 - Implementarea sistemelor de fișiere
- ▶ Curs 13 - Securitatea sistemului
- ▶ Quizz-uri curs
- Curs extra - Android
- Curs extra - Virtualizare
- Curs extra - Sincronizarea proceselor
- Note de curs

Teme

- ▶ Tema Asistenți - Guardian process
- Constații
- Git. Indicații folosire GitLab

```
#include "common.h"

/* virtual memory mapping encapsulation; initialized by vm_alloc
typedef struct vm_map {
    w_ptr_t start;
    w_handle_t ram_handle;
    w_handle_t swap_handle;
} vm_map_t;

/* initialize and cleanup library -- consider exception handler
w_boolean_t vmsim_init(void);
w_boolean_t vmsim_cleanup(void);

/**
 * allocate a virtual memory zone and corresponding RAM and swap
 *
 * map is to be filled with start address and handles to RAM and
 */
w_boolean_t vm_alloc(w_size_t num_pages, w_size_t num_frames, vm_

/*
 * free space previously allocated with vm_alloc
 * start is the start address of the previously allocated area
 *
 * implementation has to close handles corresponding to RAM and
 */
w_boolean_t vm_free(w_ptr_t start);
```

- Funcția `vmsim_init` realizează inițializarea bibliotecii. În cadrul funcției se va realiza, în general, înregistrarea *page fault* handler-ului sub forma unei rutine pentru tratarea semnalului SIGSEGV sau a unui handler de excepție.
- Funcția `vmsim_cleanup` realizează închiderea bibliotecii. În cadrul funcției se va realiza, în general, deînregistrarea *page fault* handler-ului.
- Funcția `vm_alloc` va alocă o zonă de memorie virtuală nemapată și va crea fișierele de suport (fișierul RAM și fișierul pentru spațiul de swap). Funcția primește ca argument numărul de pagini virtuale (`num_pages`) și numărul de pagini fizice (`num_frames`) pentru alocarea spațiului virtual și trunchierea fișierelor la dimensiunea necesară. Funcția întoarce pointer-ul din spațiul de adresă către zona de memorie virtuală alocată și handler-ele celor două fișiere în cadrul unei structuri `struct vm_map`. Handler-ele fișierelor vor fi folosite în cadrul suitei de test.
- Funcția `vm_free` eliberează zona de memorie virtuală mapată cu `vm_alloc` și închide și șterge fișierele asociate acesteia. Funcția primește ca argument pointer-ul către zona de memorie virtuală alocată anterior din spațiul de adresă al procesului. Implementarea trebuie să realizeze intern maparea între acest pointer și fișierele aferente pentru ca toate resursele să fie închise la apelul `vm_free`.



Trebuie să respectați prototipurile funcțiilor exportate de bibliotecă, incluzând aici tipul parametrilor. Recomandăm să nu modificați structura `vm_map`.

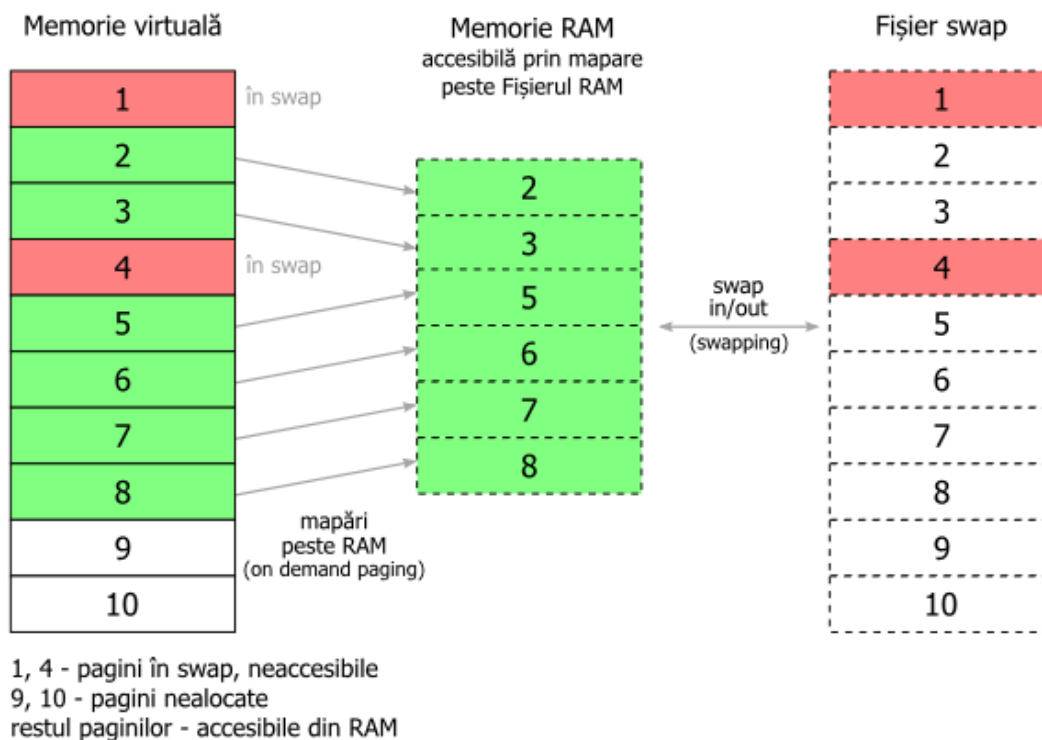
Imagine

Imaginea de mai jos prezintă "fluxul" paginilor prin memorie și o **posibilă** stare pentru o zonă dată.

- Indicații generale teme
- Tema 1 Multi-platform Development
- Tema 2 Mini-shell
- Tema 3 Memorie virtuală
- Tema 4 Planificator de threaduri
- Tema 5 Server web asincron

Table of Contents

- Tema 3 Memorie virtuală
 - Obiectivele temei
 - Recomandări
 - Enunț
 - Interfața bibliotecii
 - Imagine
 - Precizări/recomandări pentru implementare
 - Precizări pentru Linux
 - Precizări pentru Windows
 - Testare
 - Depunctări
 - Resurse de suport
 - FAQ
 - Suport, întrebări și clarificări



Această imagine este doar principială; starea descrisă nu este o stare precisă în care se poate găsi o zonă de memorie în timpul sau ca urmare a execuției testelor.

Precizări/recomandări pentru implementare

- Implementarea *page fault* handler-ului se realizează prin intermediul unei rutine pentru tratarea semnalului SIGSEGV sau a unui handler de excepție.
- Pentru a implementa logica de *demand paging* și cea de *swapping* trebuie să interceptați page fault-urile produse în momentul unui acces nevalid la o zonă de memorie. La interceptarea page fault-urilor, folosiți modificări succesive ale drepturilor paginilor pentru a implementa logica temei. La început, paginile nu au niciun drept (`PROTECTION_NONE`).
- Modificările realizate asupra memoriei virtuale nu se reflectă imediat în fișierul asociat memoriei RAM (memoria RAM fiind mapată peste acest fișier). De aceea, atunci când se dorește verificarea memoriei RAM (în teste), trebuie realizată sincronizarea paginilor din memorie cu fișierul. **Nu** este nevoie să realizați sincronizarea între memorie și fișier. Testul va realiza acest lucru, la nevoie printr-un apel la funcțiile de sincronizare a datelor în fișierele mapate.
- Nu se impune nicio restricție de nume asupra fișierelor ce reprezintă memoria RAM, respectiv spațiul de swap. Recomandăm să folosiți API-ul specific de generare a numelor de fișiere (`mkstemp`, `GetTempFileName`).
- Consultați header-ele și sursele din cadrul [arhivelor cu teste](#) pentru macro-uri și funcții utile.
- Pentru a determina dimensiunea unei pagini, folosiți funcția `w_get_page_size` definită în fișierul header `common.h`.
- Pentru implementare, puteți pleca de la următorul set de enumerări/structuri:

`helpers.h`

```

enum page_state {
    STATE_IN_RAM,
    STATE_IN_SWAP,
    STATE_NOT_ALLOC
};

struct frame;

/* handle pages (virtual pages) */
struct page_table_entry {
    enum page_state state;
    enum page_state prev_state;
    w_boolean_t dirty;
    w_prot_t protection;
    w_ptr_t start;
    struct frame *frame; /* NULL in case page is not mapped */
};

/* handle frames (physical pages) */
struct frame {
    w_size_t index;
    /* "backlink" to page_table_entry; NULL in case of free frame */
    struct page_table_entry *pte;
};

```

- Structurile de mai sus reprezintă doar un punct de pornire. Se pot adăuga alte câmpuri sau structuri specifice modului în care biblioteca va gestiona zonele de memorie.

Precizări pentru Linux

- Pentru gestiunea memoriei virtuale folosiți funcțiile `mmap`, `munmap` și `mprotect`.
- Pentru interceptarea accesului nevalid la o zonă de memorie va trebui să interceptați semnalul `SIGSEGV` folosind apeluri din familia `sigaction`.
 - Veți înregistra un handler în câmpul `sa_sigaction` al structurii `struct sigaction`.
 - Pentru a determina adresa care a generat page fault-ul folosiți câmpul `si_addr` din cadrul structurii `siginfo_t`.
- Pentru eliberarea/evacuarea unei pagini, demapați acea pagină (pagina este mapată peste o "pagină" din fișerul RAM) și remapați-o la aceeași adresă dar ca mapare anonimă (`MAP_ANONYMOUS`) și fără protecție (`PROT_NONE`).
- Pentru generarea de fișiere temporare, puteți folosi apelul `mkstemp`.
- Puteți folosi funcțiile wrapper declarate și implementate în `common.h` și `common_lin.c`.
 - Puteți extinde fișierele cu funcții wrapper proprii.
- Tema se va rezolva folosind doar funcții POSIX. Se pot folosi de asemenea și funcțiile de citire/scriere cu formatare (`scanf/printf`), funcțiile de alocare/eliberare de memorie (`malloc/free`) și funcțiile de lucru cu șiruri de caractere (`strcat, strdup` etc.)
- Pentru partea de I/O se vor folosi doar funcții POSIX. De exemplu, funcțiile `fopen, fread, fwrite, fclose` nu trebuie folosite; în locul acestora folosiți `open, read, write, close`.

Precizări pentru Windows

- API-ul oferit de Windows diferă de cel oferit de Linux; există funcții dedicate de gestiune a memoriei virtuale (`VirtualAlloc`, `VirtualFree`) și alte funcții de gestiunea a fișierelor mapate (`CreateFileMapping`, `MapViewOfFile`, `MapViewOfFileEx`, `UnmapViewOfFile`).
- Alocarea spațiului de memorie virtuală pentru o zonă se realizează folosind `VirtualAlloc`. Un apel `VirtualFree` trebuie să elibereze întreg

spațiul de memorie rezervat de un apel [VirtualAlloc](#) corespunzător. Întrucât dorim ca operațiile de alocare/dezalocare să aibă loc la nivel de pagină, recomandăm următoarea secvență de acțiuni:

- alocarea întregului spațiului necesar folosind [VirtualAlloc](#);
- eliberarea spațiului ocupat folosind [VirtualFree](#);
- alocarea consecutivă a paginilor (pagină cu pagină), la **adrese fixe**, începând cu adresa de start obținută la alocarea anterioară a întregului spațiu de memorie, folosind [VirtualAlloc](#);
- Un exemplu de utilizare a mai multor apeluri [VirtualAlloc](#) pentru pagini consecutive găsiți [aici](#).
- Pentru maparea unei pagini de memorie virtuală de la o **adresă fixă** peste un fișier folosiți funcția [MapViewOfFileEx](#).
 - Va trebui să demapați pagina de memorie virtuală (folosind [VirtualFree](#)) și să realizați noua mapare;
 - Pentru operația inversă, de demapare a unei pagini virtuale mapate peste un fișier, folosiți funcția [UnmapViewOfFile](#);
 - Un exemplu de utilizare a [MapViewOfFileEx](#) găsiți [aici](#).
- Pentru interceptarea acceselor nevalide la zone de memorie (general protection fault), va trebui să folosiți vectori de excepție; aceștia permit înregistrarea, respectiv deînregistrarea unui handler care să fie rulat la apariția unei excepții (acces nevalid). Folosiți apelurile [AddVectoredExceptionHandler](#)/[RemoveVectoredExceptionHandler](#).
 - Pentru obținerea adresei care a generat excepția (fault-ul, accesul nevalid), folosiți valoarea `ExceptionInformation[1]`, câmp al structurii [ExceptionRecord](#).
- Pentru generarea de fișiere temporare, puteți folosi apelul [GetTempFileName](#).
- Puteți folosi funcțiile wrapper declarate și implementate în `common.h` și `common_win.c`.
 - Puteți extinde fișierele cu funcții wrapper proprii.
- Tema se va rezolva folosind doar funcții Win32. Se pot folosi de asemenea și funcțiile de citire/scriere cu formatare (`scanf/printf`), funcțiile de alocare/eliberare de memorie (`malloc/free`) și funcțiile de lucru cu șiruri de caractere (`strcat, strcpy` etc.).
- Pentru partea de I/O se vor folosi doar funcții Win32. De exemplu, funcțiile `open, read, write, close` nu trebuie folosite; în locul acestora folosiți `CreateFile, ReadFile, WriteFile, CloseHandle`.

Testare

- Corectarea temelor se va realiza automat cu ajutorul unor suite de teste publice:
 - teste Linux – [tema3-checker-lin.zip](#)
 - teste Windows – [tema3-checker-win.zip](#)
- Indicații despre utilizarea suitei de teste se găsesc în fișierul `README` din cadrul arhivelor.
- Pentru evaluare și corectare, tema va fi uploadată folosind [interfața vmchecker](#).
- În urma compilării temei trebuie să rezulte o bibliotecă shared-object (pe Linux) denumită `libvmsim.so` sau o bibliotecă dinamică (pe Windows) denumită `libvmsim.dll`.
- Suita de teste conține un set de teste. Trecerea unui test conduce la obținerea punctajului aferent acestuia.
 - În urma rulării testelor, se va acorda, în mod automat, un punctaj total. Punctajul total maxim este de 95 de puncte, pentru o temă care trece toate testele. La acest punctaj se adaugă 5 puncte din oficiu.
 - Cele 100 de puncte corespund la 10 puncte din cadrul notei finale.
- **Testul 0** din cadrul checker-ului temei verifică automat coding style-ul surselor voastre. Ca referință este folosit [stilul de coding din kernelul Linux](#). Acest test valorează 5 puncte din totalul de 100. Pentru mai multe informații despre un cod de calitate citiți [pagina de recomandări](#).



Înainte de a [uploada](#) tema, asigurați-vă că implementarea voastră trece testele pe [mașinile virtuale](#). Dacă apar



probleme în rezultatele testelor, acestea se vor reproduce și pe [vmchecker](#).

Depunctări

- Pot exista penalizări în caz de întârzieri sau pentru neajunsuri de implementare sau de stil.
- Urmăriți penalizările precizate în cadrul [listei de depunctări](#)
- Suplimentar, se pot aplica depunctările:
 - `-0.5` – folosirea unei structuri de dimensiune statică pentru lista zonelor de memorie alocate folosind `vm_alloc`.
- În cazuri excepționale (e.g. tema trece testele, însă implementarea este defectuoasă sau incompletă) se pot aplica depunctări suplimentare celor menționate mai sus.

Resurse de suport

- Cursuri
 - [Curs 5 - Gestiunea memoriei](#)
 - [Curs 6 - Memoria virtuală](#)
- Laboratoare
 - [Laborator 4 - Semnale](#)
 - [Laborator 6 - Memoria virtuală](#)
- Operating System Concepts – Chapter 8 – Main Memory
- Operating System Concepts – Chapter 9 – Virtual Memory
- [Header-ul cu interfața bibliotecii](#)
- Teste
 - [Teste Linux](#)
 - [Teste Windows](#)
- [Interfața de upload vmchecker](#)
- [Lista de discuții a cursului](#)
- [Utilizarea vectorilor de excepție \(Windows\)](#)

Resursele temei se găsesc și în repo-ul [so-assignments](#) de pe GitHub. Repo-ul conține și un script Bash care vă ajută să vă creați un repository privat pe instanța de [GitLab](#) a facultății. Urmăriți indicațiile din README și de pe [pagina de Wiki dedicată pentru git](#).





În plus, responsabilii de teme se pot uita mai rapid pe [GitLab](#) la temele voastre în cazul în care aveți probleme/bug-uri. Este mai ușor să primiți suport în rezolvarea problemelor implementării voastre dacă le oferiți responsabililor de teme acces la codul sursă pe [GitLab](#).

Dacă ați folosit [GitLab](#) pentru realizarea temei, indicați în README link-ul către repository. Asigurați-vă că responsabilii de teme au drepturi de citire asupra repo-ului vostru.

FAQ

- **Q:** Tema se poate face în C++?
 - **A:** Nu.
- **Q:** Avem voie să folosim fișiere (sursă, header) prezente în arhiva de test?
 - **A:** Da, vă încurajăm să faceți acest lucru.
- **Q:** Avem voie să modificăm header-ul temei (`vm_sim.h`) sau alte headere prezente?
 - **A:** Da, dacă aceste modificări vă sunt utile în realizarea temei. Dar aceste modificări nu vor fi vizibile în cadrul testului – sursele testului sunt cele prezente în arhiva de test. **Nu** faceți modificări ale interfeței însă (adică ale semnăturii funcțiilor). Biblioteca pe care o

veți obține trebuie să expună aceleași funcții. Puteți adăuga noi funcții sau structuri dar nu modificați funcțiile sau structurile existente.

- **Q:** Dacă în implementare am folosit fișiere din cadrul testelor, trebuie să le mai includ în arhiva temei?
 - **A:** Da, trebuie să includeți în arhiva voastră toate fișierele necesare pentru a compila biblioteca.
- **Q:** Există cazuri în care funcția  FlushViewOfFile, apelată din `w_sync_mapping` iese cu eroare. Este acest lucru o problemă?
 - **A:** Situația nu constituie neapărat o problemă.
 - Funcția `w_sync_mapping` parcurge întregul spațiu de adrese virtuale (pagini virtuale) și execută flush/sync pe fiecare pagină fără a ține cont dacă sunt mapate sau nu peste fișierul de RAM.
 - Paginile mapate peste fișierul de RAM vor fi flushed/synced, în timp ce, pentru paginile nemapate, funcția  FlushViewOfFile se întoarce cu eroare.

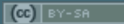
Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți [lista de discuții](#) sau [canalul de IRC](#).

so/teme/tema-3.txt · Last modified: 2017/04/26 23:29 by ioana_elena.ciornei

 Old revisions

 Media Manager  Back to top

 CHIMERIC DE   