

## Tema 3 Loader de Executabile



- Data publicare: **14.04.2022**
- Deadline: **27.04.2022, ora 23:55**
- Deadline hard: **30.04.2022, ora 23:55**

### Obiectivele temei

- Aprofundarea modului în care un executabil este încărcat și rulat de Sistemul de Operare.
- Obținerea de deprinderi pentru lucru cu excepții de memorie pe sistemele Linux și Windows.
- Aprofundarea API-ului Linux și Windows de lucru cu spațiul de adrese, memorie virtuală și *demand paging*.

### Recomandări

- Înainte de a începe implementarea temei este recomandată acomodarea cu noțiunile și conceptele specifice, precum:
  - spațiu de adresă
  - drepturi de acces la pagină
  - formatul fișierelor executabile
  - *demand paging*
  - *page fault*
  - maparea de fișiere în spațiu de adresă – *file mapping*
- Urmăriți resursele descrise în secțiunea [Resurse de suport](#).

### Enunț

Să se implementeze sub forma unei biblioteci partajate/dinamice un **loader de fișiere executabile** în format ELF pentru Linux și PE pentru Windows. Loader-ul va încărca fișierul executabil în memorie pagină cu pagină, folosind un mecanism de tipul *demand paging* – o pagină va fi încărcată doar în momentul în care este nevoie de ea. Pentru simplitate, loader-ul va rula doar executabile statice - care nu sunt link-ate cu biblioteci partajate/dinamice.

Pentru a rula un fișier executabil, loader-ul va executa următorii pași:

- Își va inițializa structurile interne.
- Va parsa fișierul binar - pentru a face asta aveți la dispoziție în scheletul temei un parser de fișiere ELF pe Linux și PE pentru Windows. Găsiți mai multe detalii în secțiunea care descrie [interfața parserului de executabile](#).
- Va rula prima instrucțiune a executabilului (*entry point-ul*).
  - de-a lungul execuției, se va genera câte un *page fault* pentru fiecare acces la o pagină nemapată în memorie;
- Va detecta fiecare acces la o pagină nemapată, și va verifica din ce segment al executabilului face parte.
  - dacă nu se găsește într-un segment, înseamnă că este un acces invalid la memorie – se rulează handler-ul default de *page fault*;
  - dacă *page fault-ul* este generat într-o pagină deja mapată, atunci se încearcă un acces la memorie nepermis (segmentul respectiv nu are permisiunile necesare) – la fel, se rulează handler-ul default de *page fault*;
  - dacă pagina se găsește într-un segment, și ea încă nu a fost încă mapată, atunci se mapează la adresa aferentă, cu permisiunile acelui segment;
  - Veti folosi funcțiile `mmap` (Linux) și `MapViewOfFileEx` (Windows) pentru a aloca memoria virtuală în cadrul procesului.
  - Pagina trebuie mapată **fix** la adresa indicată în cadrul segmentului.

### Interfața bibliotecii

Interfața de utilizare a bibliotecii loader-ului este prezentată în cadrul fișierul header `loader.h`. Acesta conține funcții de inițializare a loader-ului (`so_init_loader`) și de executare a binarului (`so_execute`).

```
loader.h
/*
 * initializes the Loader
 */
int so_init_loader(void);

/*
 * runs an executable specified in the path
 */
int so_execute(char *path, char *argv[]);
```

- Funcția `so_init_loader` 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 `so_execute` realizează parsarea binarului specificat în `path` și rularea primei instrucțiuni (*entry point*) din executabil.

### Interfața parser

Pentru a ușura realizarea temei, vă punem la dispoziție în scheletul de cod un parser pentru ELF (Linux) și PE (Windows). Deși cele două formează diferențe, interfața pusă la dispoziție este aceeași pe ambele platforme, și se găsește în header-ul `exec_parser.h`.

```
exec_parser.h
typedef struct so_seg {
    /* virtual address */
    uintptr_t vaddr;
    /* size inside the executable file */
    unsigned int file_size;
    /* size in memory (can be larger than file_size) */
    unsigned int mem_size;
    /* offset in file */
    unsigned int offset;
    /* permissions */
    unsigned int perm;
    /* custom data */
    void *data;
} so_seg_t;
```

### Informații generale SO

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

### Informatii SO 2021-2022

- Examen
  - Examen CA/CC 2012-2013
  - Examen CA/CC 2013-2014
  - Examen CA/CC 2014-2015
  - Examen CA/CC 2015-2016
  - Examen CA/CB/CC 2016-2017
  - Examen CA/CB/CC 2017-2018
  - Examen CA/CB/CC 2018-2019
  - Examen CA/CB/CC 2019-2020
  - Examen CA/CB/CC 2020-2021
- Anunțuri
- Calendar
- Echivalări teme
- Distincții
- Karma Awards
- SO Need to Know
- Reguli generale și notare
- Orar și împărțire pe semigrupe

### Laboratoare

- Resurse
  - **Windows - Tips&Tricks**
    - Tutorial Visual Studio
    - Linia de comandă în Windows
    - C/SO Tips
    - Macro-ul DIE
    - GDB
    - Function Hooking and Windows Dll Injection
    - IPC
    - Online
    - 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 - Gestiona memoria
  - Laborator 06 - Memoria virtuală
  - Laborator 07 - Profiling & Debugging
  - Laborator 08 - Threaduri Linux
  - Laborator 09 - Threaduri Windows
  - Laborator 10 - Operații IO avansate - Windows
  - Laborator 11 - Operații IO avansate - Linux
  - Laborator 12 - Implementarea sistemelor de fișiere

### Curs

- Capitol 01: Introducere
- Capitol 02: Interfața sistemului de fișiere
- Capitol 03: Procese
- Capitol 04: Planificarea execuției, IPC
- Capitol 05: Gestiona memoriei
- Capitol 06: Memoria virtuală
- Capitol 07: Analiza executabilelor și proceselor
- Capitol 08: Securitatea memoriei
- Capitol 09: Fie de execuție
- Capitol 10: Sincronizare
- Capitol 11: Dispozitive de intrare/iesire
- Capitol 12: Implementarea sistemelor de fișiere
- Capitol 13: Networking în sistemul de operare
- Capitol 14: Analiza performanței

### Teme

- Tema Asistenți - Guardian process

```

typedef struct so_exec {
    /* base address */
    uintptr_t base_addr;
    /* address of entry point */
    uintptr_t entry;
    /* number of segments */
    int segments_no;
    /* array of segments */
    so_seg_t *segments;
} so_exec_t;

/* parse an executable file */
so_exec_t *so_parse_exec(char *path);

/*
 * start an executable file, previously parsed in a so_exec_t structure
 * (jumps to the executable's entry point)
 */
void so_start_exec(so_exec_t *exec, char *argv[]);

```

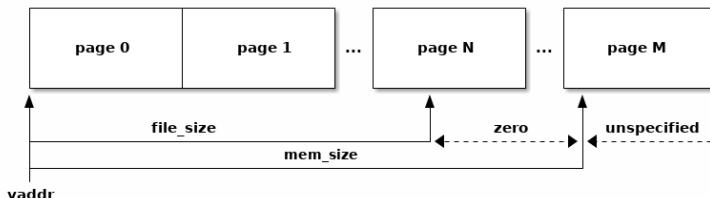
Interfața de parser pune la dispoziție două funcții:

- `so_parse_exec` - parsează executabilul și întoarce o structură de tipul `so_exec_t`. Aceasta poate fi folosită în continuare pentru a identifica segmentele executabilului și atributile lui.
- `so_start_exec` - pregătește environment-ul programului și începe execuția lui.
  - Începând din acest moment, se vor executa *page fault-uri* pentru fiecare acces de pagină nouă/nemapată.

Structurile folosite de interfață sunt:

- `so_exec_t` - descrie structura executabilului:
  - `base_addr` - indică adresa la care ar trebui încărcat executabilul
  - `entry` - adresa primei instrucțiuni execute de către executabil
  - `segments_no` - numărul de segmente din executabil
  - `segments` - un vector (de dimensiunea `segments_no`) care conține segmentele executabilului
- `so_seg_t` - descrie un segment din cadrul executabilului
  - `vaddr` - adresa la care ar trebui încărcat segmentul
  - `file_size` - dimensiunea în cadrul fișierului a segmentului
  - `mem_size` - dimensiunea ocupată de segment în memorie; dimensiunea segmentului în memorie poate să fie mai mare decât dimensiunea în fișier (spre exemplu pentru segmentul `bss`); în cazul acesta, diferența între spațiul din memorie și spațiul din fișier, trebuie zerofizată
  - `offset` - offsetul din cadrul fișierului la care începe segmentul
  - `perm` - o mască de biți reprezentând permisiunile pe care trebuie să le aibă paginile din segmentul curent
    - `PERM_R` - permisiuni de citire
    - `PERM_W` - permisiuni de scriere
    - `PERM_X` - permisiuni de execuție
  - `data` - un pointer opac pe care îl puteți folosi să atașați informații proprii legate de segmentul curent (spre exemplu, puteți stoca aici informații despre paginile din segment deja mapate)

În imaginea de mai jos aveți o reprezentare grafică a unui segment.



### Precizări/recomandări pentru implementare

- Implementarea *page fault* handler-ului se realizează prin intermediul unei `SIGSEGV` sau a unui `handler` de excepție.
- Pentru a implementa logica de *demand paging* trebuie să interceptați *page fault*-urile produse în momentul unui acces nevalid la o zonă de memorie. La interceptarea *page fault*-urilor, tratați-o corespunzător, în funcție de segmentul din care face parte:
  - dacă nu este într-un segment cunoscut, rulați *handler*-ul default;
  - dacă este într-o pagină ne-mapată, mapați-o în memorie, apoi copiați din segmentul din fișier datele;
  - dacă este într-o pagină deja mapată, rulați *handler*-ul default (intrucât este un acces ne-permis la memorie);
- Paginile din două segmente diferite nu se pot suprapune.
- Dimensiunea unui segment nu este aliniată la nivel de pagină; memoria care nu face parte dintr-un segment nu trebuie tratată în niciun fel – comportamentul unui acces în cadrul aceliei zone este nedefinit.
- **NU** se vor depunca resursele leak-uite datorită faptului că programul se termină înainte de a avea posibilitatea să fie eliberate:
  - structurile rezultante în urma parsării executabilului (`so_exec_t` și `so_seg_t`);
  - structurile alocate de voi și stocate în `field`-ul `data` al unui segment;
  - paginile mapate în memorie în urma execuției *on-demand*.
- Pentru implementare vă recomandăm să porniți de la scheletele puse la dispoziție de echipa de Sisteme de Operare pentru `Linux` și `Windows`.

### Precizări pentru Linux

- Pentru gestiunea memoriei virtuale folositi funcțiile `mmap`, `munmap` și `protect`.
- Pentru interceptarea accesului nevalid la o zonă de memorie ce trebuie să interceptați semnalul `SIGSEGV` folosind apeluri din familia `sigaction`.
  - Văți 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`.
- În momentul în care este accesată o pagină nouă din cadrul unui segment, mapați pagina în care s-a generat *page fault*-ul, folosind `MAP_FIXED`, apoi copiați în pagină datele din executabil
- 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 siruri 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,`

- Contestări
- Git. Indicații folosire GitLab
- Indicații generale teme
- Hackathon SO
- Tema 1 Multi-platform Development
- Tema 2 Bibliotecă stdio
- Tema 3 Loader de Executabile
- Tema 4 Planificator de threaduri
- Tema 5 Server web asincron

### Table of Contents

- Tema 3 Loader de Executabile
  - Obiectivele temei
  - Recomandări
  - Enunț
  - Interfață bibliotecii
  - Interfață parser
  - Precizări/recomandări pentru implementare
  - Precizări pentru Linux
  - Precizări pentru Windows
  - Testare
    - Depunctări
  - Resurse de suport
  - FAQ
  - Suport, întrebări și clarificări

- MapViewOfFile,MapViewOfFileEx, UnmapViewOfFile.
- Pentru alocarea unei pagini la o **adresă fixă** folosiți funcția **MapViewOfFileEx** sau **VirtualAlloc**.
- Deși dimensiunea unei pagini pe Windows este 4096 de bytes, funcțiile **VirtualAlloc/MapViewOfFileEx** pot aloca doar cu granularitatea de 65536 bytes, ceea ce înseamnă că adresele alocate trebuie să fie multiplu de 65536 (0x10000), nu 4096 (0x1000). Din această cauză veți considera că pe Windows dimensiunea paginii este 0x10000.
- Pentru interceptarea acceselor nevalide la zone de memorie (general protection fault), va trebui să folosiți vectori de excepție; acestia 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**.
- 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 siruri 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**.
- Pentru a rula executabile în format PE, acestea trebuie linkate cu biblioteca Windows kernel.dll – pentru asta, înainte de a rula executabilul, trebuie rezolvate toate simbolurile din această bibliotecă; scheletul pus la dispozitiv face asta, prin urmare, pe Windows primele *page fault*-uri generate vor fi de scriere, nu de execuție de cod.

## Testare

- Pentru testare vom folosi doar binare linkate static (fără dependențe externe).
- Corectarea temelor se va realiza automat cu ajutorul unor suite de teste publice:
  - **Teste Linux**
  - **Teste Windows**
- Pentru a rula loader-ul în afara testelor, puteți folosi binarul de test (**so\_test\_prog**) din cadrul scheletului.
- 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 **libso\_loader.so** sau o bibliotecă dinamică (pe Windows) denumită **so\_loader.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 teste 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**.
- În cazuri exceptionale (e.g. tema trece teste, însă implementarea este defectuoasă sau incompletă) se pot aplica depunctări suplimentare celor menționate mai sus.

## Resurse de suport

- Cursuri
  - **Curs 5 - Gestiona memoria**
  - **Curs 6 - Memoria virtuală**
  - **Curs 7 - Securitatea memoriei**
- Laboratoare
  - **Laborator 4 - Semnale**
  - **Laborator 5 - Gestiona memoria**
  - **Laborator 6 - Memoria virtuală**
- Operating System Concepts – Chapter 8 – Main Memory
- Operating System Concepts – Chapter 9 – Virtual Memory
- Teste
  - **Teste Linux**
  - **Teste Windows**
- Schelet
  - Directorul **3-loader** din **repo-ul** de pe Github
- **Interfața de upload vmchecker**
- **forumul temei**
- **Utilizarea vectorilor de excepție (Windows)**
- **Formatul fișierelor executable pe Linux (ELF)**
- **Formatul fișierelor executable pe Windows (PE)**

Resursele temei se găsesc și în **repo-ul so**, directorul 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, responsabilități 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 oferți responsabilități de teme acces la codul sărșă pe **GitLab**.

Dacă ati folosit **GitLab** pentru realizarea temei, indicați în README link-ul către repository. Asigurați-vă că responsabilități de teme au drepturi de citire asupra repo-ului vostru.

## FAQ

- **Q:** Tema se poate face în C++?
  - **A:** Nu.
- **Q:** Avemvoie să folosim fișiere (sursă, header) prezente în arhiva de test?
  - **A:** Da, vă încurajăm să faceți acest lucru.
- **Q:** Avemvoie să modificăm header-ul temei (**loader.h**) sau alte headere prezente?
  - **A:** Nu.
- **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.

## Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți **forumul temei**.

Orice întrebare pe forum trebuie să conțină o descriere cât mai clară a eventualei probleme. Întrebări de forma: "Nu merge X. De ce?" fără o descriere mai amănunțită vor primi un răspuns mai greu. Înainte să postați o întrebare pe forum căti și celealte întrebări(dacă există) pentru a vedea dacă întrebarea voastră a fost deja adresată sub o altă formă(in cazul în care răspunsul din partea echipei vine mai greu este mai rapid să căutați voi deja printre întrebările existente).



**ATENȚIE** să nu postați imagini cu părți din soluția voastră pe forumul pus la dispoziție sau orice alt canal public de comunicație. Dacă veți face acest lucru, vă asumăți răspunderea dacă veți primi copiat pe temă.

Logged in as: Bogdan Mihai TUDORACHE (108609) (bogdan.tudorache99)

so/teme/tema-3.txt · Last modified: 2022/04/14 20:51 by ionut.mihalache1506

[Old revisions](#)

[Media Manager](#) [Manage Subscriptions](#) [Back to top](#)

[CC BY-SA](#) [OHMERIC DE](#) [W3C CSS](#) [DOKUMENIT](#) [GET FIREFOX](#) [RSS XML FEED](#) [W3C XHTML 1.0](#)