Loader de Executabile

- **Deadline**: **24.11.2022, ora 23:55**

Obiectivele temei

- Aprofundarea modului în care un executabil este încărcat și rulat de Sistemul de Operare.
- Obținerea de deprinderi pentru lucrul cu excepții de memorie pe un sistem de operare.
- Utilizarea API-ului Linux 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 spațu 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 [https://en.wikipedia.org/wiki/Executable_and_Linkable_Format] pentru Linux. 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 dispozitie în scheletul temei un parser de fișiere ELF pe Linux. 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;
- Veți folosi funcția mmap [http://www.kernel.org/doc/man-pages/online/pages/man2/mmap.2.html] (Linux) 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 loaderului(`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 [http://www.kernel.org/doc/man-pages/online/pages/man2/sigaction.2.html] sau a unui handler de excepție [http://msdn.microsoft.com/en-us/library/ms681420(v=vs.85).aspx].
- 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).

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;

typedef struct so_exec {
    /* base adress */
    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[]);
```

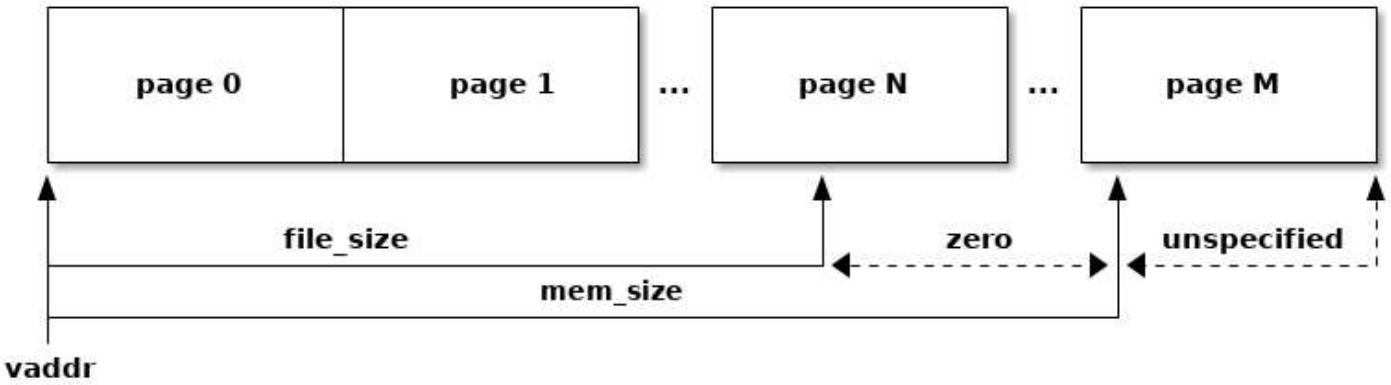
Interfața de parser pune la dispoziție două funcții:

- `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 atributele 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 folosit de interfață sunt:

- `so_exec_t` - descrie structura executabilului:
  - `base_addr` - indică adresa la care ar trebui încărcat executabilul
  - `entry` - adresa primei instrucțiuni executate 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 zeroizată
  - `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 srcriere
    - `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 rutine pentru tratarea semnalului SIGSEGV [http://www.kernel.org/doc/man-pages/online/pages/man2/sigaction.2.html].
- 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 (întrucât 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 acelei zone este nedefinit.
- **NU** se vor depuncta resursele leak-uite datorită faptului că programul se termină înainte de a avea posibilitatea să fie eliberate:
  - structurile rezultate 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.

Precizări

- Pentru gestiunea memoriei virtuale folosiți funcțiile mmap [http://www.kernel.org/doc/man-pages/online/pages/man2/mmap.2.html], munmap [http://www.kernel.org/doc/man-pages/online/pages/man2/munmap.2.html] și mprotect [http://www.kernel.org/doc/man-pages/online/pages/man2/mprotect.2.html].
- Pentru interceptarea accesului nevalid la o zonă de memorie va trebui să interceptați semnalul SIGSEGV folosind apeluri din familia sigaction [http://www.kernel.org/doc/man-pages/online/pages/man2/sigaction.2.html].
  - 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`.
- 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 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`.
- Un exemplu de asociere a unui handler de tratare a unui semnal este prezentat mai jos:

```
#include <signal.h>
...

/* SIGUSR2 handler */
static void usr2_handler(int signum) {
    /* actions that should be taken when the signal signum is received */
    ...
}

int main(void) {
    struct sigaction sa;

    memset(&sa, 0, sizeof(sa));

    sa.sa_flags   = SA_RESETHAND;   /* restore handler to previous state */
    sa.sa_handler = usr2_handler;
    sigaction(SIGSEGV, &sa, NULL);

    return 0;
}
```

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 [https://github.com/systems-cs-pub-ro/so/tree/2022-2023/teme/assignments/1-loader/checker-lin]
- 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 [https://elf.cs.pub.ro/vmchecker/ui].
- În urma compilării temei trebuie să rezulte o bibliotecă shared-object (pe Linux) denumită `libso_loader.so`.
- 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 [https://www.kernel.org/doc/Documentation/process/coding-style.rst]. Acest test nu puncte din totalul de 100 însă ajută la o organizare mai bună a codului. Pentru mai multe informații (dacă este nevoie) despre un cod de calitate citiți pagina de recomandări [http://ocw.cs.pub.ro/courses/so/laboratoare/resurse/c_tips].

Înainte de a uploada [https://vmchecker.cs.pub.ro] tema, asigurați-vă că implementarea voastră trece testele pe mașina virtuală de linux [http://ocw.cs.pub.ro/courses/so/info/mv]. Dacă apar probleme în rezultatele testelor, acestea se vor reproduce și pe vmchecker [https://vmchecker.cs.pub.ro] sau pe orice alt mediu de testare solicitat.

Depunctări

- Pot exista penalizări în caz de întârzieri sau pentru neajunsuri de implementare sau de stil.
- Urmăriți penalizările precizate în cadrul listei de depunctări [http://ocw.cs.pub.ro/courses/so/teme/general#lista_depunctari]
- În cazuri excepționale (e.g. tema trece testele, însă implementarea este defectuoasă sau incompletă) se pot aplica depunctări suplimentare celor menționate mai sus.
- În general nu vă concentrați pe depunctări, nu este scopul nostru să vă dăm note mici, ci doar să vă atenționăm unde lucrurile pot fi făcute mai bine fără a aplica depunctări care să vă afecteze semnificativ nota finală.

Resurse de suport

- Partea de curs și laborator care prezintă partea de gestiune a memoriei și memorie virtuală
- Resursele de mai jos sunt mai vechi și nu trebuie consultate în detaliu pentru rezolvarea temei. Anumite concepte pot să vă fie de folos însă părțile de care aveți nevoie din ele nu reprezintă o parte semnificativă din rezolvarea temei
  - Cursuri (Old but gold)
    - Gestiunea memoriei [http://ocw.cs.pub.ro/courses/so/cursuri/curs-05]
    - Memoria virtuală [http://ocw.cs.pub.ro/courses/so/cursuri/curs-06]
    - Securitatea memoriei [http://ocw.cs.pub.ro/courses/so/cursuri/curs-07]
  - Laboratoare (Old but gold)
    - Creare makefile și biblioteci [https://ocw.cs.pub.ro/courses/so/laboratoare/laborator-01]]
    - Semnale [http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-04]
    - Gestiunea memoriei [http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-05]
    - Memoria virtuală [http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-06]
  - Operating System Concepts – Chapter 8 – Main Memory
  - Operating System Concepts – Chapter 9 – Virtual Memory
- Teste
  - Teste Linux [https://github.com/systems-cs-pub-ro/so/tree/2022-2023/teme/assignments/1-loader/checker-lin]
- Schelet
  - Directorul https://github.com/systems-cs-pub-ro/so/tree/2022-2023/teme/assignments/1-loader [https://github.com/systems-cs-pub-ro/so/tree/2022-2023/teme/assignments/1-loader] din repo-ul de pe Github [https://github.com/systems-cs-pub-ro/so/tree/2022-2023/teme/assignments]
- Interfața de upload vmchecker [https://elf.cs.pub.ro/vmchecker/ui]
- forumul temei [https://curs.upb.ro/2022/mod/forum/view.php?id=76307]
- Formatul fișierelor executabile pe Linux (ELF) [https://en.wikipedia.org/wiki/Executable_and_Linkable_Format]

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 (`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 [https://curs.upb.ro/2022/mod/forum/view.php?id=76307].

Orice intrebare 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 citiți și celelalte întrebări(dacă există) pentru a vedea dacă întrebarea voastră a fost deja adresată sub o altă formă(în 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 asumați răspunderea dacă veți primi copiat pe temă.