Virtual Memory Allocator

**Responsabili**

* [Radu Nichita](mailto:radunichita99@gmail.com)
* [Maria Sfîrăială](mailto:maria.sfiraiala@gmail.com)
* [Andreea Drehuță](mailto:andreea.dre02@gmail.com)
* Data publicării: **22.03.2023 22:00:00**
* Deadline **HARD**:  **12.04.2023 23:55:00**

Actualizări

* Adăugare mesaje de eroare (INVALID\_ALLOC\_BLOCK): **28.03.2023**
* Adăugare clarificări arenă: **24.03.2023**

Obiective

* Aprofundarea cunoștințelor în utilizarea limbajului C.
* Implementarea și utilizarea structurii de date [listă dublu înlănțuită](https://en.wikipedia.org/wiki/Doubly_linked_list).
* Familiarizarea cu implementarea unei **structuri de date generice**.

Introducere

Studenții de anul întâi au aflat de la colegii mai mari de o materie interesantă pe care urmează să o aibă în anul 2, faimoasa Observarea Sistemelor de Operare (cu acronimul OSO). Fiind curioși, au început deja investigația, și pe lângă meme-uri, debate-uri și cereri de prelungire a deadline-urilor temelor, au găsit și concepte inedite, precum [memoria virtuală](https://en.wikipedia.org/wiki/Virtual_memory).

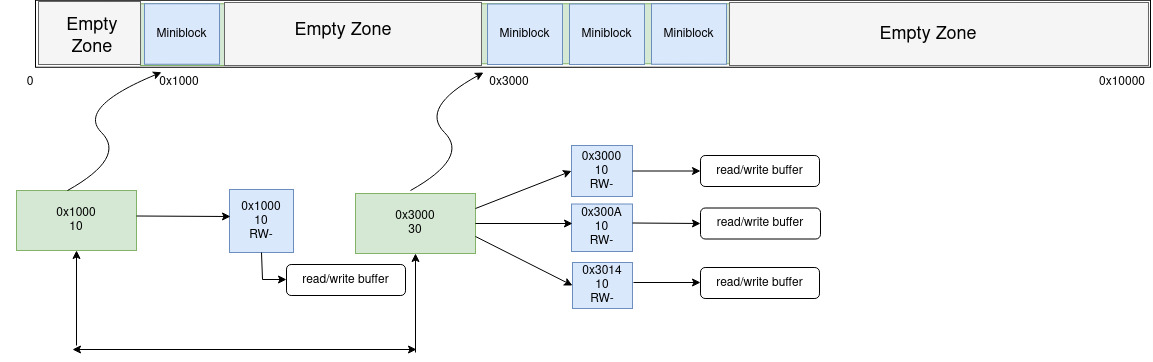
Bineînțeles, studenții sunt niște ființe deloc leneșe, așa că s-au apucat deja de prima temă, un alocator de memorie virtuală, pe care vi-l arată și vouă, extaziați. Scopul vostru este să le mai micșorați avântul și să implementați împreună cu frații voștrii de suferință această funcționalitate a sistemului de operare, folosind **exclusiv** liste dublu înlănțuite. Pe parcurs, nu veți avea nevoie de cunoștințe de Sisteme de Operare, dar, dacă într-adevăr, vă pasionează subiectul, vizitați [pagina oficială a cursului](https://open-education-hub.github.io/operating-systems/).

Cerință

Un alocator de memorie are rolul de a **rezerva** memorie, la nivel de bibliotecă, tradițional prin apeluri precum malloc() sau calloc(). Acestea marchează ca fiind folosite anumite zone de memorie dintr-un pool de bytes prealocat, numit arenă. De asemenea, alocatorul de memorie se ocupă și cu eliberarea zonelor rezervate, apelul de bibliotecă aferent fiind free().

Alocatorul de memorie virtuală pe care îl veți implementa va avea următoarele funcționalități (sau API):

* un buffer ascuns, pe care operațiile directe sunt interzise, numit **kernel buffer** sau **arena**; acesta este un vector generic, ce reține informația alocată de voi
* o **listă dublu înlănțuită** ce reține zonele alocate din arenă; aceasta memorează și zonele alocate adiacent, reprezentate, la rândul lor, ca **liste dublu înlănțuite**



Astfel, lista dublu înlănțuită principală reține **block-urile** alocate, pe când listele dublu înlănțuite din aceasta rețin **miniblock-urile** consecutive. O structură posibilă pentru **block** ar arăta în felul următor:

{

uint64\_t start\_address; *// adresa de început a zonei, un indice din arenă*

size\_t size; *// dimensiunea totală a zonei, suma size-urilor miniblock-urilor*

void\* miniblock\_list; *// lista de miniblock-uri adiacente*

} block\_t;

O structură posibilă pentru **miniblock** ar arăta în felul următor:

{

uint64\_t start\_address; *// adresa de început a zonei, un indice din arenă*

size\_t size; *// size-ul miniblock-ului*

uint8\_t perm; *// permisiunile asociate zonei, by default RW-*

void\* rw\_buffer; *// buffer-ul de date, folosit pentru opearțiile de read() și write()*

} miniblock\_t;

Comenzi posibile (40p)

Input-ul este oferit de la stdin, iar output-ul la stdout, respectând formatul următor:

* ALLOC\_ARENA <dimensiune\_aren>
  + Se alocă un buffer contiguu de dimensiune ce va fi folosit pe post de **kernel buffer** sau **arenă**. Alocarea este pur virtuală, adică acest buffer este folosit **doar** pentru a **simula** existența unui spațiu fizic aferent zonelor de memorie înlănțuite.
* DEALLOC\_ARENA
  + Se eliberează arena alocată la începutul programului. Cum alocarea arenei a fost făcută virtual, această comandă rezultă în eliberarea tuturor resurselor folosite, precum lista de block-uri și listele de miniblock-uri asociate acestora.
* ALLOC\_BLOCK <adresă\_din\_arenă> <dimensiune\_block>
  + Se marchează ca fiind rezervată o zonă ce începe la adresa <adresă\_din\_arenă> în **kernel buffer** cu dimensiunea de <dimensiune\_block>
  + Dacă nicio adresă din zona de memorie [adresă\_din\_arenă, adresă\_din\_arenă + dimensiune) nu a mai fost alocată anterior și **nu** există alocate zonele de memorie [x, adresă\_din\_arenă - 1) și [adresă\_din\_arenă + dimensiune + 1, y], unde x < adresă\_din\_arenă - 1 și y > adresă\_din\_arenă + dimensiune + 1, atunci se inserează un nou block în lista de zone alocate.
  + Dacă nicio adresă din zona de memorie [adresă\_din\_arenă, adresă\_din\_arenă + dimensiune) nu a mai fost alocată anterior și **există** alocate zonele de memorie [x, adresă\_din\_arenă - 1) sau [adresă\_din\_arenă + dimensiune + 1, y], unde x < adresă\_din\_arenă - 1 și y > adresă\_din\_arenă + dimensiune + 1, atunci se șterge block-ul/block-urile adiacente din lista de zone alocate și se adaugă la lista internă de miniblock-uri a noii zone contigue de memorie. Cu alte cuvinte, zonele adiacente din memorie vor fi **mereu** în același block.
* FREE\_BLOCK <adresă\_din\_arenă>
  + Se eliberează un miniblock.
  + Dacă se eliberează unicul miniblock din cadrul unui block, atunci block-ul este la rândul său eliberat.
  + Dacă se eliberează un miniblock de la începutul/sfârșitul acestuia, atunci structura block-ului nu se modifică.
  + Dacă se eliberează un miniblock din mijlocul block-ului, atunci acesta va fi împărțit în două block-uri distincte.
* READ <adresă\_din\_arenă> <dimensiune>
  + Se afișează <dimensiune> bytes începând cu adresa <adresă\_din\_arenă>, iar la final \n.
  + Dacă block-ul nu conține <dimensiune> bytes începând cu adresa <adresă\_din\_arenă>, se va afișa **“Warning: size was bigger than the block size. Reading <size> characters.\n”** și se vor afișa datele disponibile.
* WRITE <adresă\_din\_arenă> <dimensiune> <date>
  + Se scriu <dimensiune> bytes din <date> la adresa <adresă\_din\_arenă>.
  + Dacă <date> nu conține <dimensiune> bytes pe același rând, se va citi în continuare, până la atingerea dimensiunii stabilite.
  + Dacă block-ul nu conține <dimensiune> bytes începând cu adresa <adresă\_din\_arenă>, se va afișa **“Warning: size was bigger than the block size. Writing <size> characters.\n”**.
* PMAP
  + Tradițional, apelul [pmap()](https://man7.org/linux/man-pages/man1/pmap.1.html), este folosit pentru a vizualiza zonele de memorie utilizate de un proces. Printre acestea se numără .text, .bss, .data, .rodata, etc, însă voi veți avea de implementat o funcționalitate mai facilă.
  + Se afișează informații despre block-urile și miniblock-urile existente.
  + Formatul permisiunilor este **RWX**. Dacă vreuna dintre aceste permisiuni lipsește, ea va fi înlocuită cu -.

Dacă nu ați implementat bonusul, permisiunile miniblock-urilor vor fi mereu **RW-**!

Formatul de afișare pentru PMAP este următorul:

Total memory: x

Free memory: y

Number of allocated blocks: z

Number of allocated miniblocks: q

Block 1 begin

Zone: address1 - address2

Miniblock 1:\t\taddress1\t\t-\t\taddress1a\t\t| permissions

Miniblock 2:\t\taddress1a\t\t-\t\taddress1b\t\t| permissions

.

.

.

Miniblock n:\t\taddress1x\t\t-\t\taddress2\t\t| permissions

Block 1 end

Block 2 begin

Zone: address3 - address4

Miniblock 1:\t\taddress3\t\t-\t\taddress3a\t\t| permissions

Miniblock 2:\t\taddress3a\t\t-\t\taddress3b\t\t| permissions

.

.

.

Miniblock m:\t\taddress3y\t\t-\t\taddress4\t\t| permissions

Block 2 end

.

.

.

Block z begin

.

.

.

Block z end

Tratarea erorilor (40p)

La primirea comenzilor, pot apărea erori de input. Pe acestea, voi trebuie să le tratați corespunzător, prin afișarea unui mesaj sugestiv:

* INVALID\_ALLOC\_BLOCK
  + Dacă adresa de început a blocului ce se dorește a fi alocat depășește dimensiunea arenei, se va afișa **“The allocated address is outside the size of arena\n”**.
  + Dacă adresa de final a blocului ce se dorește a fi alocat depășește dimensiunea arenei, se va afișa **“The end address is past the size of the arena\n”**.
  + Dacă există cel puțin o adresă din zona de memorie [adresă\_din\_arenă, adresă\_din\_arenă + dimensiune) care a fost alocată anterior, se va afișa **“This zone was already allocated.\n”**.
* INVALID\_ADDRESS\_FREE
  + Dacă se încearcă eliberarea zonei de memorie asociate unei adrese invalide (nu a fost alocată sau nu reprezintă o adresă de început de miniblock),se afișează **“Invalid address for free.\n”**.
* INVALID\_ADDRESS\_READ
  + Dacă se încearcă citirea de la o adresă invalidă (nu a fost alocată), se afișează **“Invalid address for read.\n”**.
* INVALID\_ADDRESS\_WRITE
  + Dacă se încearcă scrierea la o adresă invalidă (nu a fost alocată), se afișează **“Invalid address for write.\n”**.
* INVALID\_COMMAND
  + Dacă este introdusă o comandă inexistentă sau dacă numărul de parametri al acesteia este incorect, se afișează **“Invalid command. Please try again.\n”** și se va trece la următoarea comandă.

Bonus (20p)

* MPROTECT <adresă\_din\_arenă> <noua\_permisiune>
  + Schimbă permisiunile zonei care începe la <adresă\_din\_arenă> de dimensiune <dimensiune> din default-ul RW- în <noua\_permisiune>.
  + Noua permisiune este dată ca mască de biți (reținute ca string-uri), astfel:
    - **PROT\_NONE**: Memoria nu poate fi accesată deloc; în octal, masca este reprezentată prin 0
    - **PROT\_READ**: Memoria poate fi citită; în octal, masca este reprezentată prin 4
    - **PROT\_WRITE**: Memoria poate fi scrisă; în octal, masca este reprezentată prin 2
    - **PROT\_EXEC**: Memoria poate fi executată; în octal, masca este reprezentată prin 1
  + Astfel, o zonă cu permisiunile RW-, are reținut, în octal, flag-ul 6 , care este și valoarea default a unei zone nou alocate.
  + Va trebui să citiți string-urile primite de la tastatură și să le interpretați numeric. Pentru permisiuni multiple, acestea vor fi înlănțuite prin operatorul |.
  + Prin schimbarea permisiunilor, în cadrul operațiile de READ și WRITE trebuie să se verifice dacă există permisiuni pentru aplicarea acestora. Dacă acestea nu există se va afișa, după caz, **“Invalid permissions for read.\n”** sau **“Invalid permissions for write.\n”**.
  + Dacă se încearcă schimbarea permisiunilor zonei de memorie asociate unei adrese invalide (nu a fost alocată sau nu reprezintă o adresă de început de miniblock), se afișează **“Invalid address for mprotect.\n”**.

Exemplu

Fie input-ul:

ALLOC\_ARENA 65536

ALLOC\_BLOCK 4096 10

ALLOC\_BLOCK 12288 10

ALLOC\_BLOCK 12308 10

PMAP

ALLOC\_BLOCK 12298 10

PMAP

WRITE 4096 26 Observ sisteme de operare

READ 4096 14

FREE\_BLOCK 12298

MPROTECT 12308 PROT\_NONE

WRITE 12308 24 Sper să nu iau SEGFAULT

PMAP

DEALLOC\_ARENA

Output-ul corespunzător acestei secvențe de operații este:

Total memory: 0x10000 bytes

Free memory: 0xFFE2 bytes

Number of allocated blocks: 3

Number of allocated miniblocks: 3

Block 1 begin

Zone: 0x1000 - 0x100A

Miniblock 1: 0x1000 - 0x100A | RW-

Block 1 end

Block 2 begin

Zone: 0x3000 - 0x300A

Miniblock 1: 0x3000 - 0x300A | RW-

Block 2 end

Block 3 begin

Zone: 0x3014 - 0x301E

Miniblock 1: 0x3014 - 0x301E | RW-

Block 3 end

Total memory: 0x10000 bytes

Free memory: 0xFFD8 bytes

Number of allocated blocks: 2

Number of allocated miniblocks: 4

Block 1 begin

Zone: 0x1000 - 0x100A

Miniblock 1: 0x1000 - 0x100A | RW-

Block 1 end

Block 2 begin

Zone: 0x3000 - 0x301E

Miniblock 1: 0x3000 - 0x300A | RW-

Miniblock 2: 0x300A - 0x3014 | RW-

Miniblock 3: 0x3014 - 0x301E | RW-

Block 2 end

Warning: size was bigger than the block size. Writing 10 characters.

Warning: size was bigger than the block size. Reading 10 characters.

Observ sis

Invalid permissions for write.

Total memory: 0x10000 bytes

Free memory: 0xFFE2 bytes

Number of allocated blocks: 3

Number of allocated miniblocks: 3

Block 1 begin

Zone: 0x1000 - 0x100A

Miniblock 1: 0x1000 - 0x100A | RW-

Block 1 end

Block 2 begin

Zone: 0x3000 - 0x300A

Miniblock 1: 0x3000 - 0x300A | RW-

Block 2 end

Block 3 begin

Zone: 0x3014 - 0x301E

Miniblock 1: 0x3014 - 0x301E | ---

Block 3 end

Acest exemplu corespunde figurii de mai sus. Asigurați-vă că urmăriți exemplul și figura în paralel, pentru o mai bună înțelegere a comenzilor. Astfel:

* Primul PMAP afișează 3 block-uri cu câte un miniblock fiecare. Block-urile 2 și 3 nu sunt încă unite deoarece există un spațiu de 10 bytes nealocat între zonele celor două.
* Al doilea PMAP este aplicat imediat după alocarea block-ului care ar trebui să unească cele 2 block-uri menționate anterior într-unul singur. Acest lucru se întâmplă și rezultă 2 block-uri, primul cu un singur miniblock, iar al doilea cu trei miniblock-uri.
* Se încearcă scrierea a 25 de caractere, dar cum dimensiunea miniblock-ului în care se realizează WRITE este de doar 10, se afișează mesajul de warning și se rețin în buffer-ul miniblock-ului primele 10 caractere.
* Se încearcă citirea a 14 caractere din cadrul miniblock-ului în care s-a scris la comanada precedentă, dar cum acesta conține 10 caractere, se afișează mesajul de warning și doar cei 10 bytes reținuți.
* Prin MPROTECT, permisiunile ultimului miniblock sunt schimbate în **PROT\_NONE**, așadar operația de WRITE următoare eșuează și se afișează mesajul de eroare.
* Al treilea PMAP afișează rezultatul operației de FREE\_BLOCK, care a despărțit, din nou, block-ul cu trei miniblock-uri în două. De asemenea, permisiunile ultimului miniblock existent au devenit **—**, semnificând că pe zona respectivă nu este posibilă nicio operație.

README (10p)

* Fișierul de README va avea numele **README** sau **README.md** și va fi pus în **rădăcina arhivei**. În acest fișier veți detalia implementarea voastră, ce vi s-a părut interesant și care a fost cea mai dificilă parte a temei.
* Găsiți [aici](https://ocw.cs.pub.ro/courses/_media/sd-ca/teme/readme-skel.md) un template de README. Recomandăm folosirea formatului [Markdown](https://www.markdownguide.org/basic-syntax/) în scrierea acestuia.

Coding style (10p)

Mențiuni

* Se garantează faptul că operația de READ va citi din zone scrise anterior cu WRITE. Cu alte cuvinte o operație validă de READ va avea **mereu** loc după o operație validă de WRITE.
* Indexarea se face **de la 0** pentru orice tip de listă.
* Alocarea memoriei se va realiza **dinamic**. Alocarea statică a bufferelor a căror dimensiune nu se cunoaște la compile-time, atrage dupa ea depunctări.
* Se cere ca structura de date folosită să fie **generică**. Pentru implementare lipsită de genericitate, se vor pierde **20p din totalul temei**.
* O structură de date **generică** are rolul de a nu duplica (pe cât posibil) cod permițând adăugarea în cadrul ei a **oricărui tip de date**. (hint: void \*)
* Exemplu: Vedeți în cadrul scheletului de lab unde a trebuit să implementați exact această situație. (hint: data\_size)
* **Nu** este indicată utilizarea **variabilelor globale**. Utilizarea acestora aduce pierderea a **20p din totalul temei**.
* Eliberarea memoriei se va verifica folosind utilitarul Valgrind. O temă ce conține memory leaks va atrage după sine punctaj de **0p pe testul respectiv**.
* Este **obligatorie** construirea unei implementări care folosește conceptul de listă în listă.
* Orice tentativă de alocare a unui vector liniar, contiguu și de simulare a operațiilor aplicate arenei pe acesta este **interzisă**. Arena există doar la nivel conceptual, task-urile temei aplicându-se pe lista dublu înlănțuită de block-uri, respectiv miniblock-uri.
* Nerespectarea acestor constrângeri va conduce la **notarea temei cu 0 puncte**.

Checker

Scheletul poate fi găsit [aici](https://github.com/sd-pub/sd-2023-homeworks).

Temele vor fi trimise pe [vmchecker](https://vmchecker.cs.pub.ro/ui/#SD). **Atenție!** Temele trebuie trimise în secțiunea **Structuri de Date (CA)**.

Arhiva trebuie să conțină:

* sursele **.c și .h**
* fișier **Makefile** cu două reguli:
  + regula **build**: în urma căreia se generează un executabil numit **vma**
  + regula **run\_vma**: în urma căreia va rula executabilul **vma**
  + regula **clean** care şterge executabilul si fișierele obiect
* Compilarea trebuie sa se realizeze cu flagurile **-Wall -Wextra -std=c99**
* fișier **README** care să conțină detalii despre implementarea temei