Ľubomír Dlhý, ID: 102897

SPRÁVCA PAMÄTE

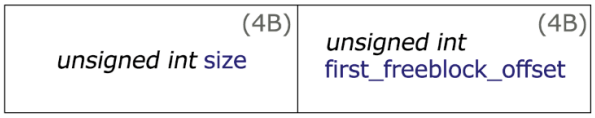
Ľubomír Dlhý, ID: 102897

POUŽITÝ ALGORITMUS

V mojej implementácii prideľovania pamäte som využil explicitný zoznam voľných blokov, ktorý je však zoradený vzostupne podľa veľkosti blokov. Zoznam je pospájaný v oboch smeroch a na posun medzi blokmi využíva offsety počítané od ukazovateľa na začiatok spravovanej pamäte.

V programe využívam 3 typy hlavičiek:

- hlavička spravovanej pamäte (8 Bajtov) – obsahuje informáciu o veľkosti spravovanej pamäte (4 Bajty) a offset na prvý voľný blok (4 Bajty)

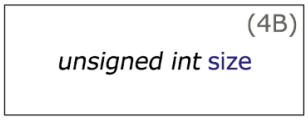



```
typedef struct hlavicka_pamate {
    uint size;
    uint first_freeblock_offset;
} HLAVICKA_PAMATE;
```

- hlavička voľného bloku (12 Bajtov) - obsahuje informáciu o veľkosti voľného bloku (4 Bajty), offset na predchádzajúci voľný blok (4 Bajty) a offset na nasledujúci voľný blok (4 Bajty)


```
typedef struct volny_blok {
    uint size;
    uint offset_prev;
    uint offset_next;
} VOLNY_BLOK;
```

- hlavička alokovaného bloku (4 Bajty) – obsahuje informáciu o veľkosti alokovaného bloku



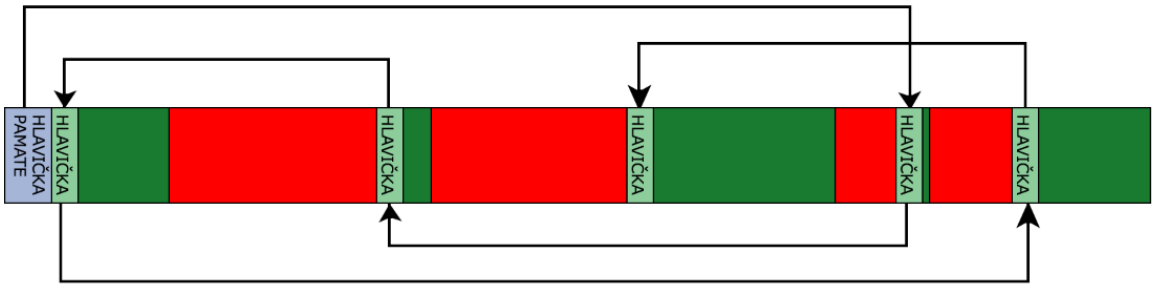

```
typedef struct alokovany_blok {
    uint size;
} ALOKOVANY_BLOK;
```

Keďže je zoznam zoradený podľa veľkosti voľných blokov, v alokácii stačí použiť vyhľadávaciu metódu First-Fit na to, aby sme našli najvhodnejší voľný blok a minimalizovali tak externú fragmentáciu.

Ľubomír Dlhý, ID: 102897

ALOKÁCIA PAMÄTE

Keď užívateľ požiada o alokovanie bloku v pamäti, prehľadáme zoznam voľných blokov a pokúsime sa nájsť voľný blok, ktorého veľkosť je dostatočne veľká na alokáciu. Veľkosť voľného bloku musí byť aspoň rovná súčtu požadovanej veľkosti na alokáciu dát a veľkosti hlavičky alokovaného bloku (4B). Keďže je náš explicitný zoznam usporiadaný podľa veľkosti voľných blokov, postačí nám prehľadávacia metóda first-fit, aby sme našli najlepší možný voľný blok a minimalizovali tak fragmentáciu. Voľné bloky, ktoré aj po alokácii majú dostatok miesta (minimálne 12B na hlavičku voľného bloku), rozdelíme a novovytvorený menší voľný blok vložíme naspäť do zoznamu. Ak voľný blok nemá dostatok miesta na rozdelenie, alokuje sa celý voľný blok, to aj v prípade, že po alokovaní zostane miesto (maximálne 11Byteov). Toto zarovnávanie napomáha pri internej fragmentácii.



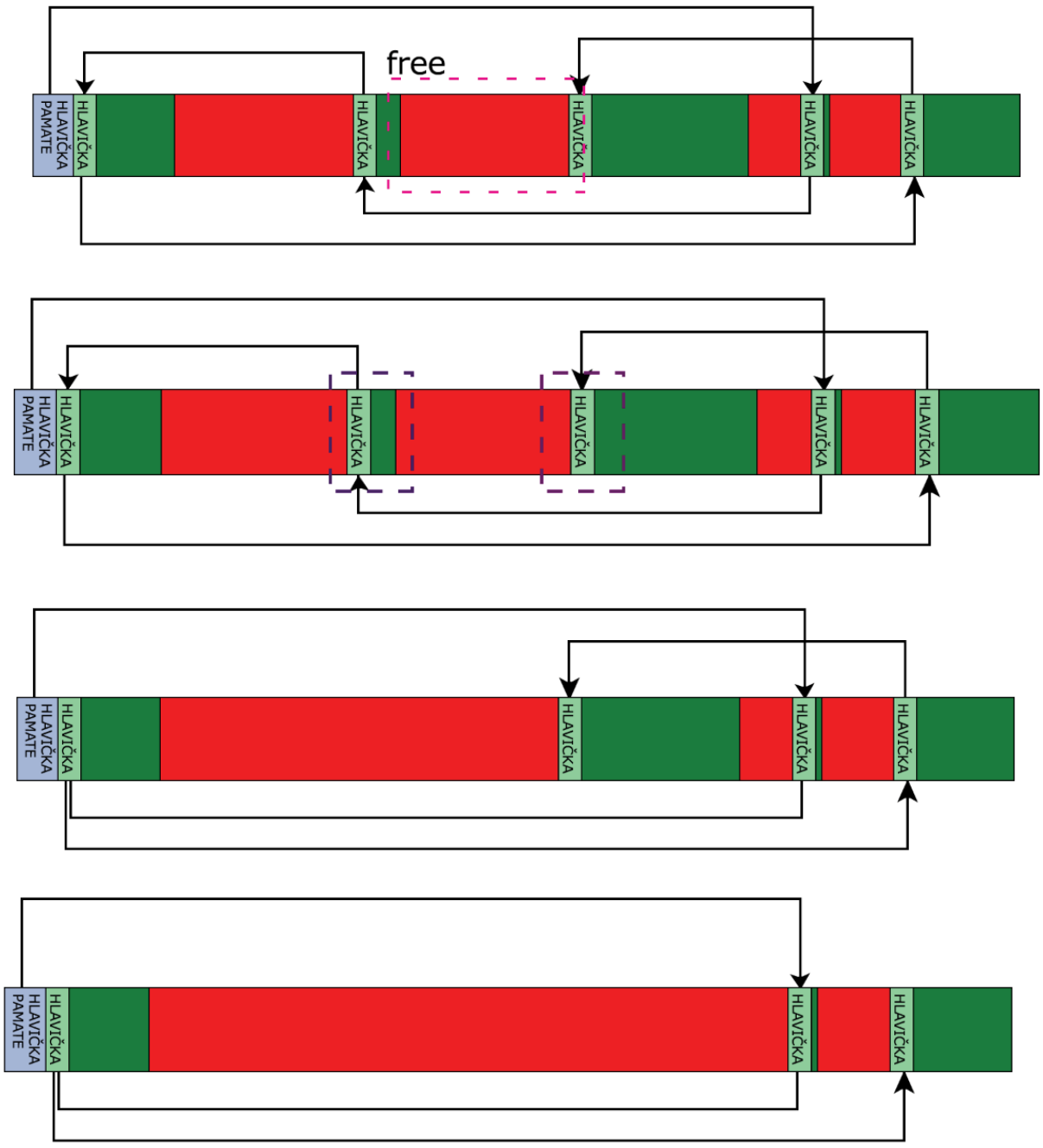
Zložitosť:
Časová zložitosť je v najhoršom prípade 2N, kde N je počet voľných blokov. (First-fitom na koniec zoznamu, alokuje blok, rozdelí a potom usporiada blok do zoznamu).
V najlepšom prípade je časová zložitosť 1 a to v prípade, že je zoznam voľných blokov prázdny.
Všeobecne je časová zložitosť prideľovania pamäte O(n), kde n je počet voľných blokov.

Pamäťová zložitosť je v najhoršom prípade N+11 a v najlepšom prípade N+4, kde N je požadovaná veľkosť alokácie.
Všeobecne je teda pamäťová zložitosť O(n), kde n je požadovaná veľkosť alokácie.

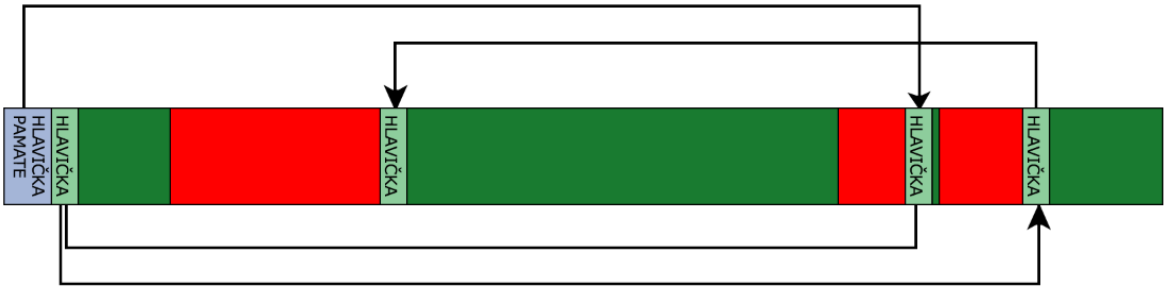
Ľubomír Dlhý, ID: 102897

UVOĽNOVANIE PAMÄTE

Pri uvoľňovaní alokovaného bloku najprv zisťujeme, či s alokovaným blokom susedí voľný blok. To zistíme prehľadaním zoznamu voľných blokov a porovnávaním adries ukazovateľov. Ak alokovaný blok susedí s voľným blokom, susedný voľný blok spojíme s alokovaným a odstránime ho z explicitného zoznamu. Takto minimalizujeme fragmentáciu. Následne pôvodne alokovaný blok pridáme do usporiadaného zoznamu voľných blokov.



Ľubomír Dlhý, ID: 102897



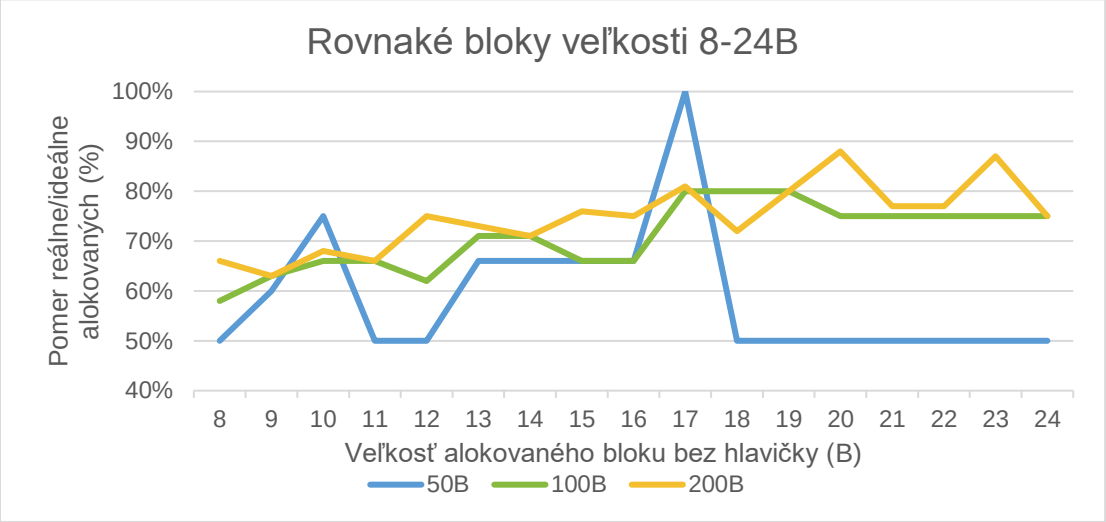
Zložitosť:
Časová zložitosť je v najhoršom prípade 2N, kde N je počet voľných blokov. (prejde celým zoznamom pri hľadaní susedných blokov, usporiadanie bloku do zoznamu).
V najlepšom prípade je časová zložitosť 1 a to v prípade, že je zoznam voľných blokov prázdny.
Všeobecne je časová zložitosť prideľovania pamäte O(n), kde n je počet voľných blokov.

Ľubomír Dlhý, ID: 102897

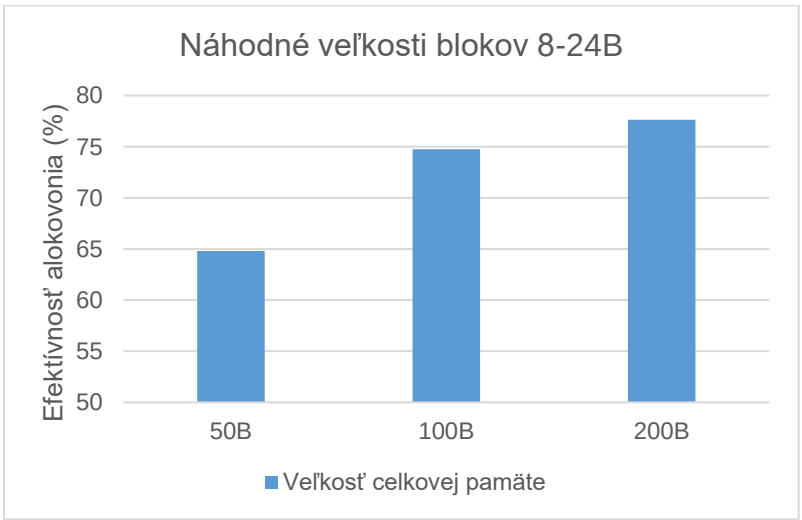
TESTOVANIE

Na testovanie funkčnosti a efektívnosti programu som spravil testovacie funkcie. Pre rovnaké veľkosti blokov som spravil test `testSingleNumber()`, ktorý vyalokoval celú spravovanú pamäť blokom rovnakej veľkosti. Potom som porovnal, koľko blokov mohlo byť alokovaných, keby nepoužívame réžiu pamäte a zistil tak efektivitu.

Automatické testy som použil pri blokoch s náhodnou veľkosťou. Taktiež som vyalokoval celú spravovanú pamäť blokmi náhodnej veľkosti a vyhodnotil koľko blokov bolo alokovaných oproti ideálnemu riešeniu bez réžie. Automatické testy som spúšťal 1000-krát a spravil priemer, aby som minimalizoval odchýlku.

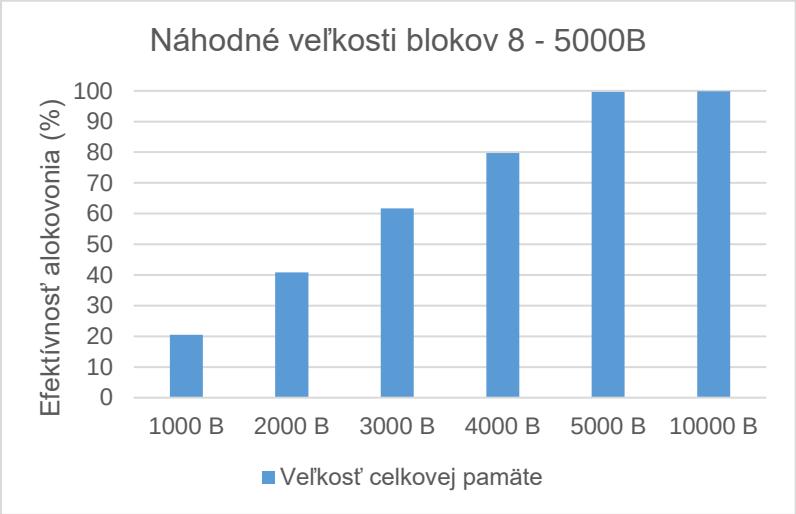


Rovnaké bloky veľkosti 8 až 24 Bajtov som vyhodnotil pre celkovú pamäť veľkosti 50 Bajtov, 100Bajtov a 200Bajtov. Efektivita alokovania bola nízka pri malej veľkosti (50B), keďže réžia pamäte tvorí značnú časť. Zväčšovaním celkovej pamäte stúpa aj efektivita alokovania.



Ľubomír Dlhý, ID: 102897

Pre náhodné veľkosti som scenár vyhodnocoval na vzorke 1000 testov. Pri blokoch náhodnej veľkosti 8-24 Bajtov som porovnal celkové veľkosti pamäte 50Bajtov, 100Bajtov a 200Bajtov. Opäť sa potvrdilo, že pri väčšej celkovej pamäti je efektivita alokovania väčšia. Konkrétne pre 200B celkovú pamäť bolo alokovaných 77,64% blokov, zatiaľ čo pri 50B iba 64,81%.



Pri celkovej pamäti veľkej veľkosti (1kb-10kb) a blokoch s veľkým rozptylom (8-5000B) je rozdiel ešte značnejší. Zatiaľ čo pri 1kb pamäti bola efektivita alokácie iba 20,45% pri 2kb bola už 40,87% a pri 5kb 99,69%.

Pre zjednodušenie debugovania a overenie správnosti fungovania programu som spravil funkciu `echoMemory()`, ktorá prešla zoznam voľných blokov a na základe informácií o voľných blokoch vypísala stav pamäte – počet voľných blokov a ich veľkosť, alokovanú pamäť. Pomocou tejto funkcie som testoval správnosť funkcie memory_free.

```
[1652828] Initiated 100 Bytes of memory.
 [  HEADER: 8B (8.00%) FREE(1 BLOCKS): 92 (92.00%) ALLOCATED: 0 (0.00%) ]

[1652836 - 1652850] Successfully allocated block of 10 Bytes. [+4B HEADER]
[1652850 - 1652864] Successfully allocated block of 10 Bytes. [+4B HEADER]
[1652864 - 1652878] Successfully allocated block of 10 Bytes. [+4B HEADER]
[1652878 - 1652892] Successfully allocated block of 10 Bytes. [+4B HEADER]
[1652892 - 1652906] Successfully allocated block of 10 Bytes. [+4B HEADER]

 [  HEADER: 8B (8.00%) FREE(1 BLOCKS): 22 (22.00%) ALLOCATED: 70 (70.00%) ]

[1652836 - 1652850] Successfully freed memory

 [  HEADER: 8B (8.00%) FREE(2 BLOCKS): 36 (36.00%) ALLOCATED: 56 (56.00%) ]

[1652864 - 1652878] Successfully freed memory

 [  HEADER: 8B (8.00%) FREE(3 BLOCKS): 50 (50.00%) ALLOCATED: 42 (42.00%) ]

 - Mergujem [1652836 - 1652850] predchadzajuci blok s [1652850 - 1652864] alokovanym.
 - Mergujem [1652836 - 1652864] alokovany blok s [1652864 - 1652878] nasledujucim.
[1652836 - 1652878] Successfully freed memory

 [  HEADER: 8B (8.00%) FREE(2 BLOCKS): 64 (64.00%) ALLOCATED: 28 (28.00%) ]
```

```
memory_init(region, 100);

char* p1 = memory_alloc(10);

char* p2 = memory_alloc(10);

char* p3 = memory_alloc(10);

char* p4 = memory_alloc(10);

char* p5 = memory_alloc(10);

memory_free(p1);
memory_free(p3);
memory_free(p2);
```

Ľubomír Dlhý, ID: 102897

ZHODNOTENIE

Zadaním projektu bolo vytvoriť vlastnú verziu implementácie prideľovania pamäte. Využil som explicitný zoznam voľných blokov, ktorý som usporiadal podľa veľkosti blokov. Vďaka čomu mi na alokáciu stačila vyhľadávacia metóda first-fit na nájdenie najvhodnejšieho bloku, čím som minimalizoval fragmentáciu. Pri rozdeľovaní blokov som využíval zarovnávanie (padding), čím som tiež znížil fragmentáciu.

Správnosť mojej metódy, funkcií memory_alloc, memory_free a memory_check som overil vo vlastných automatických testoch. V nich som ukázal, že réžia pamäte je nesmierne dôležitá pri alokácii pamäte. Moja metóda funguje najlepšie pri alokovaní veľkých blokov vo veľkej spravovanej pamäti.

V zadaní som vyjadril časovú a priestorovú zložitosť môjho riešenia, kde mi vyšlo, že asymptotická časová aj priestorová zložitosť je lineárna. Pri použití lepšej metódy – oddelených zoznamov pre rôzne veľkosti voľných blokov by sme vedeli alokáciu ešte viac zefektívniť.