SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičová 2, 842 16 Bratislava 4

Zadanie 1 – Správca pamäte

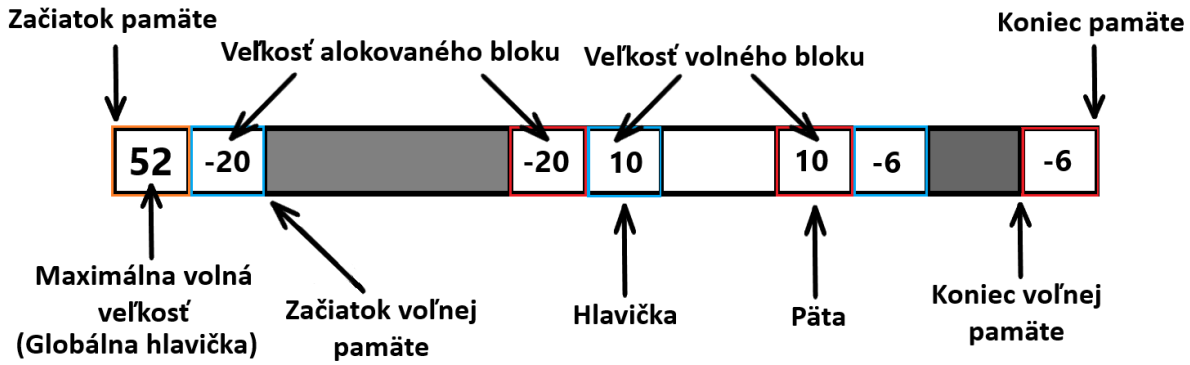
ADRIÁN VANČO
ID: 103171

Cvičenie: Utorok 18:00
8.3.2021

Úvod

Správcu pamäte som implementoval s použitím metódy implicitného zoznamu blokov pamäte. S využitím hlavičiek a pätičiek s použitím ofsetov uložených v nich ako celé číslo, ktoré využijem na prechádzanie v pamäti. Nealokovaný blok pamäte reprezentujem v hlavičke a päte ako kladné číslo veľkosti bloku bez veľkosti hlavičky a päty. Alokovaný blok pamäte reprezentujem ako záporné číslo veľkosti bloku bez veľkosti hlavičky a päty.

Názorná vizualizácia pamäte



Globálna hlavička ma veľkosť **4 Bajtov** – obsahuje maximálnu veľkosť pamäte, ktoré je možné alokovať s pridelenou pamäťou.

Hlavička ma veľkosť **4 Bajtov** – obsahuje kladné číslo veľkosti voľnej pamäte, ak je blok volný, alebo záporné číslo veľkosti voľnej pamäte ak je alokovaný.

Na **začiatok pamäte** po vykonaní funkcie memory_init bude ukazovať globálny ukazovateľ void* memory.

Funkcia memory_init

Funkcia memory_init slúži na inicializáciu spravovanej voľnej pamäte. Predpokladáme, že funkcia sa volá práve raz pred všetkými inými volaniami memory_alloc, memory_free a memory_check.

Funkcia má dva argumenty – ukazovateľ na začiatok pamäte, ktorú máme prideliť a veľkosť pamäte na správu.

```
void memory_init(void* ptr, unsigned int size);
```

V tejto funkcii sa do globálneho premennej **void* memory** uloží adresa začiatku pamäte, ktorú chceme prideliť.

```
memory = (void*)ptr;          //inicializácia ukazovatela na začiatok spravovanej pamäte
```

Následne sa na prvé **4Bajty** uloží maximálna veľkosť voľnej pamäte, to znamená že od priradenej veľkosti odrátame veľkosť globálnej hlavičky, prvej hlavičky a poslednej päty. Túto veľkosť zapíšeme aj do druhých **4Bajtov** čo je naša prvá hlavička a do posledných **4Bajtov** pamäte, čo je naša päta.

```
*memorySize = size - (3 * sizeof(int));
*memoryHead = size - (3 * sizeof(int));
*memoryFooter = size - (3 * sizeof(int));
```

Pamäť po inicializácií

```
0x00DD7284   +52   +0   +0   +0  +52   +0   +0   +0
0x00DD728C    +0   +0   +0   +0   +0   +0   +0   +0
0x00DD7294    +0   +0   +0   +0   +0   +0   +0   +0
0x00DD729C    +0   +0   +0   +0   +0   +0   +0   +0
0x00DD72A4    +0   +0   +0   +0   +0   +0   +0   +0
0x00DD72AC    +0   +0   +0   +0   +0   +0   +0   +0
0x00DD72B4    +0   +0   +0   +0   +0   +0   +0   +0
0x00DD72BC    +0   +0   +0   +0  +52   +0   +0   +0
```

Funkcia memory_alloc

Funkcia memory_alloc má poskytovať služby analogické štandardnému malloc. Teda, vstupné parametre sú veľkosť požadovaného súvislého bloku pamäte a funkcia vráti: ukazovateľ na úspešne alokovaný kus voľnej pamäte, ktorý sa vyhradil, alebo NULL, keď nie je možné súvislú pamäť požadovanej veľkosť vyhradiť.

Pri implementácií som využil metódu best fit, a teda pri alokovaný sa použije voľný blok s najbližšou veľkosťou k požadovanej. Funkcia prehľadáva pamäť od začiatku po blokoch, ak narazí na presne požadovanú veľkosť nealokovaného bloku uloží sa adresa začiatku bloku do ukazovateľa a prehľadávanie sa zastaví a ide sa na alokáciu bloku.

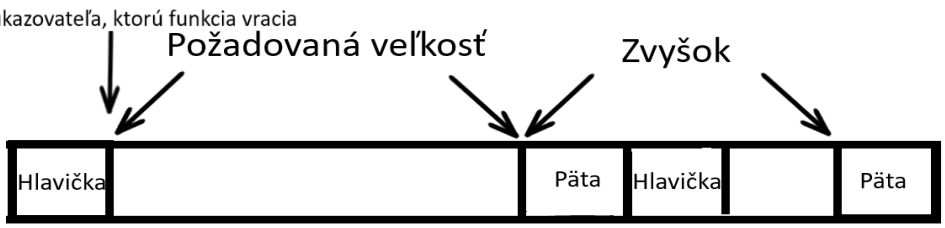
```
if (*memorySize > 0 && *memorySize == size) {

    if (memoryForAlloc) {
        memoryForAlloc = memorySize;
    }
    else {
        memoryForAlloc = memorySize;
    }
    break;
}
```

Ak veľkosť nie je rovnaká tak sa následne skontroluje či je blok alokovaný, väčší ako požadovaný a menší ako odložený ak toto spĺňa uložím novú adresu bloku.

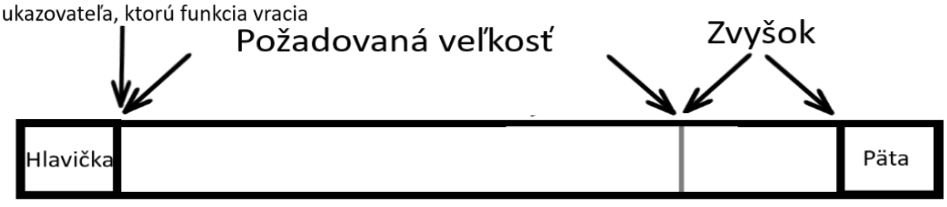
```
if (*memorySize > 0 && (unsigned)(*memorySize) > size && *memoryForAlloc > *memorySize) {
    memoryForAlloc = memorySize;
}
```

Ak sa vyhovujúci voľný blok nenašiel v pamäti nemôže sa alokovať a teda vraciame NULL. Ale ak sa našiel vyhodnotím sa či sa ho oplatí rozdeliť aby nám nevznikli fragmenty v pamäti. Rozdeliť sa ho oplatí ak sa do zvyšku po rozdelený zmestí hlavička s pätou a aspoň 1B pre alokáciu. V alokovanom bloku prehodí v hlavičke a päte hodnotu na zápornú.

```
if ((unsigned)(*memoryForAlloc) > size + 2 * sizeof(int)) {
```




```
else {
```



Na záver funkcia vracia ukazovateľ na začiatok alokovaných blokov.

Funkcia memory_free

Funkcia memory_free slúži na uvoľnenie vyhradeného bloku pamäti, podobne ako funkcia free. Funkcia vráti 0, ak sa podarilo (funkcia zbehla úspešne) uvoľniť blok pamäti, inak vráti 1. predpokladáme, že parameter bude vždy platný ukazovateľ, vrátený z predchádzajúcich volaní funkcie memory_alloc, ktorý ešte nebol uvoľnený.

Vďaka použitiu hlavičky a päty, sa pri uvoľňovaný bloky dokážu spájať v konštantnom čase. O(1).

Ako prvé funkcia zisťuje, či uvoľňovaný blok nie je na okraji pamäte. Následne podľa toho sa rozhodne kam sa môže pozrieť.

```
bool left = true;       //premenná či sa môže pozrieť na ľavo od uvolňovanej pamäte
bool right = true;      //premenná či sa môže pozrieť na pravo od uvolňovanej pamäte

//hlavička uvolnovanej pamäti je prvá hlavička v pamäti
if (memoryBegin == memorySize) {
    left = false;       //nemôže sa pozrieť do ľava
}
//päta uvolnovanej pamäti je posledná päta v pamäti
if (memoryEnd == (int*)memoryPtrEnd) {
    right = false;      //nemôže sa pozrieť do prava
}
```

Podľa hodnôt v premenných **left** a **right** vznikajú nasledujúce scenáre:

- **left** == **false** a **right** == **false**
  - nespája sa pretože je to jediný blok v pamäti, iba uvoľním.
- **left** == **true** a **right** == **false**
  - funkcia sa pozrie či blok vľavo je voľný, ak áno spoja sa a vznikne nový voľný blok s veľkosťou ľavého + uvoľňovaného + 8B, ak nie uvoľní sa iba blok.
- **left** == **false** a **right** == **true**
  - funkcia sa pozrie či blok vpravo je voľný, ak áno spoja sa a vznikne nový voľný blok s veľkosťou pravého + uvoľňovaného + 8B, ak nie uvoľní sa iba blok.
- **left** == **true** a **right** == **true**
  - funkcia sa pozrie či blok vľavo je voľný, ak áno spoja sa, ak nie blok sa uvoľní následne sa skontroluje blok vpravo ak je voľný spoja sa ak nie uvoľní sa blok.

Funkcia memory_check

Funkcia memory_check slúži na skontrolovanie, či parameter (ukazovateľ) je platný ukazovateľ, ktorý bol v nejakom z predchádzajúcich volaní vrátený funkciou memory_alloc a zatiaľ nebol uvoľnený funkciou memory_free. Funkcia vráti 0, ak je ukazovateľ neplatný, inak vráti 1.

Funkcia najprv skontroluje či je pridelená pamäť , či ukazovateľ, ktorý má program overiť nie je NULL,

```
//kontrola či pointer nieje NULL
if (memory == NULL || ptr == NULL) {
    return 0;
}
```

následne či je z rozsahu pamäte ak nie je funkcia vracia 0.

```
//kontrola či je zo spravovanej pamäte
if (ptr >= memoryStart && ptr < memoryEnd) {
```

Ak je prechádzam pamäť po blokoch a hľadám zhodu s ukazovateľom na začiatok voľnej pamäte bloku ak sa nájde zhoda overí sa či je blok alokovaný. Ak je funkcia vráti 1 ak nie 0.

Testovanie

Pre testovanie alokácie som si vytvoril 2 funkcie. test1() pre alokovanie blokov rovnakej veľkosti

a test2 pre alokáciu rôzne veľkých blokov z intervalu.

1. prideľovanie rovnakých blokov malej veľkosti (veľkosti 8 až 24 bytov) pri použití malých celkových blokov pre správcu pamäte (do 50 bytov, do 100 bytov, do 200 bytov)

```
Test 1: memory size 25, block size 8, ideal 3, allocated 1, count/ideal 33.00 %
Test 1: memory size 50, block size 8, ideal 6, allocated 2, count/ideal 33.00 %
Test 1: memory size 75, block size 8, ideal 9, allocated 4, count/ideal 44.00 %
Test 1: memory size 100, block size 8, ideal 12, allocated 6, count/ideal 50.00 %
Test 1: memory size 150, block size 8, ideal 18, allocated 9, count/ideal 50.00 %
Test 1: memory size 200, block size 8, ideal 25, allocated 12, count/ideal 48.00 %
```

```
Test 1: memory size 25, block size 14, ideal 1, allocated 0, count/ideal 0.00 %
Test 1: memory size 50, block size 14, ideal 3, allocated 2, count/ideal 66.00 %
Test 1: memory size 75, block size 14, ideal 5, allocated 3, count/ideal 60.00 %
Test 1: memory size 100, block size 14, ideal 7, allocated 4, count/ideal 57.00 %
Test 1: memory size 150, block size 14, ideal 10, allocated 6, count/ideal 60.00 %
Test 1: memory size 200, block size 14, ideal 14, allocated 8, count/ideal 57.00 %
```

```
Test 1: memory size 25, block size 23, ideal 1, allocated 0, count/ideal 0.00 %
Test 1: memory size 50, block size 23, ideal 2, allocated 1, count/ideal 50.00 %
Test 1: memory size 75, block size 23, ideal 3, allocated 2, count/ideal 66.00 %
Test 1: memory size 100, block size 23, ideal 4, allocated 3, count/ideal 75.00 %
Test 1: memory size 150, block size 23, ideal 6, allocated 4, count/ideal 66.00 %
Test 1: memory size 200, block size 23, ideal 8, allocated 6, count/ideal 75.00 %
```

2. prideľovanie nerovnakých blokov malej veľkosti (náhodné veľkosti 8 až 24 bytov) pri použití malých celkových blokov pre správcu pamäte (do 50 bytov, do 100 bytov, do 200 bytov),

```
Blocks for test 2: 10 23
Test 2: memory size 25, ideal size 10, allocated 1, count/ideal 100.00 %

Blocks for test 2: 10 23 15 16
Test 2: memory size 50, ideal size 48, allocated 2, count/ideal 66.00 %

Blocks for test 2: 10 23 15 16 12
Test 2: memory size 75, ideal size 64, allocated 3, count/ideal 75.00 %

Blocks for test 2: 10 23 15 16 12 8 12 9
Test 2: memory size 100, ideal size 96, allocated 4, count/ideal 57.00 %

Blocks for test 2: 10 23 15 16 12 8 12 9 22 20 17
Test 2: memory size 150, ideal size 147, allocated 6, count/ideal 60.00 %

Blocks for test 2: 10 23 15 16 12 8 12 9 22 20 17 17 16 24
Test 2: memory size 200, ideal size 197, allocated 9, count/ideal 69.00 %
```

3. prideľovanie nerovnakých blokov väčšej veľkosti (veľkosti 500 až 5000 bytov) pri použití väčších celkových blokov pre správcu pamäte (aspoň veľkosti 1000 bytov),

```
Blocks for test 2: 952
Test 2: memory size 500, ideal size 0, allocated 0, count/ideal 0.00 %

Blocks for test 2: 952 3149
Test 2: memory size 1000, ideal size 952, allocated 1, count/ideal 100.00 %

Blocks for test 2: 952 3149
Test 2: memory size 2000, ideal size 952, allocated 1, count/ideal 100.00 %

Blocks for test 2: 952 3149 4863
Test 2: memory size 5000, ideal size 4101, allocated 2, count/ideal 100.00 %

Blocks for test 2: 952 3149 4863
Test 2: memory size 7500, ideal size 4101, allocated 2, count/ideal 100.00 %

Blocks for test 2: 952 3149 4863 2724
Test 2: memory size 10000, ideal size 8964, allocated 3, count/ideal 100.00 %

Blocks for test 2: 952 3149 4863 2724 1197 3195 3151 545 4916 2025
Test 2: memory size 25000, ideal size 24692, allocated 9, count/ideal 100.00 %
```

4. prideľovanie nerovnakých blokov malých a veľkých veľkostí (veľkosti od 8 bytov do 50 000) pri použití väčších celkových blokov pre správcu pamäte (aspoň veľkosti 1000 bytov).

```
Blocks for test 2: 5363
Test 2: memory size 500, ideal size 0, allocated 0, count/ideal 0.00 %

Blocks for test 2: 5363
Test 2: memory size 1000, ideal size 0, allocated 0, count/ideal 0.00 %

Blocks for test 2: 5363
Test 2: memory size 2000, ideal size 0, allocated 0, count/ideal 0.00 %

Blocks for test 2: 5363
Test 2: memory size 5000, ideal size 0, allocated 0, count/ideal 0.00 %

Blocks for test 2: 5363 13991
Test 2: memory size 7500, ideal size 5363, allocated 1, count/ideal 100.00 %

Blocks for test 2: 5363 13991
Test 2: memory size 10000, ideal size 5363, allocated 1, count/ideal 100.00 %

Blocks for test 2: 5363 13991 24212
Test 2: memory size 25000, ideal size 19354, allocated 2, count/ideal 100.00 %

Blocks for test 2: 5363 13991 24212 7138
Test 2: memory size 50000, ideal size 43566, allocated 3, count/ideal 100.00 %

Blocks for test 2: 5363 13991 24212 7138 28715
Test 2: memory size 75000, ideal size 50704, allocated 4, count/ideal 100.00 %

Blocks for test 2: 5363 13991 24212 7138 28715 3173 29543
Test 2: memory size 100000, ideal size 82592, allocated 6, count/ideal 100.00 %
```

Pre testovanie memory_free a memory_check som v main funkcii vytvoril postupnú alokáciu ukazovateľov rôznej veľkosti s výpisom a následne postupným uvoľňovaným v stanovenom poradí som overil scenáre, ktoré môžu nastať pri uvoľňovaní.

Zhodnotenie

Časová zložitosť je O(n), pretože v najhoršom prípade pri alokovaní musí program prejsť celú pamäť veľkosti n. Čas na alokáciu bloku teda závisí od miesta v pamäti, na ktoré program blok alokuje.

Pamäťová zložitosť je O(n), pretože program pracuje s n-prvkovým polom.

Spájanie blokov je O(1) vďaka hlavičke a päte.

Riešenie pri malých veľkostiach pamäte sa efektivita pohybuje oproti ideálnemu riešeniu okolo 34%.

Pri väčších je to okolo 60%.