Martin Mihalovič
Streda 18:00, ID: 96969

Dátové štruktúry a algoritmy

Zadanie č. 1 – Správca pamäti

Martin Mihalovič

Martin Mihalovič
Streda 18:00, ID: 96969

Použitý prístup

V mojom programe som si vybral explicitný prístup k pamäťovému manažmentu, keďže oproti implicitnému je výkonnejší. Môj program som staval na testovaní, pri ktorom som sa primarne zameriaval na funkčnosť, výkonnosť, modularitu, neredundanciu kódu a hraničné prípady.

Použitý algoritmus

V programe som použil algoritmus Best fit, pretože mi prišiel najefektívnejší z pamäťových aspektov. Takisto som sa ho snažil optimalizovať aby bol aj výkonný a rýchly, čo si myslím, že sa mi do veľkej miery podarilo.

Aplikácia algoritmu v memory_alloc funkcii bola, že som buď prešiel celé pole voľných pamäťových blokov a na konci si vybral najvhodnejší. Alebo som počas prechádzania cez pole natrafil na úplne presný voľný blok aký potrebujem alokovať, a vtedy som už nepokračoval v prehľadávaní na koniec.

Rozhodoval som sa medzi First fit alebo Best fit, avšak algoritmus first fit mi prišiel zbytočným rozbíjaním nadmerne veľkých pamäťových blokov, ktoré môžu v prípade potreby užívateľovi chýbať…

Pri uvoľňovaní pamäte som sa zameral na fragmentáciu a tiež na ošetrenie okrajových prípadov.

Program

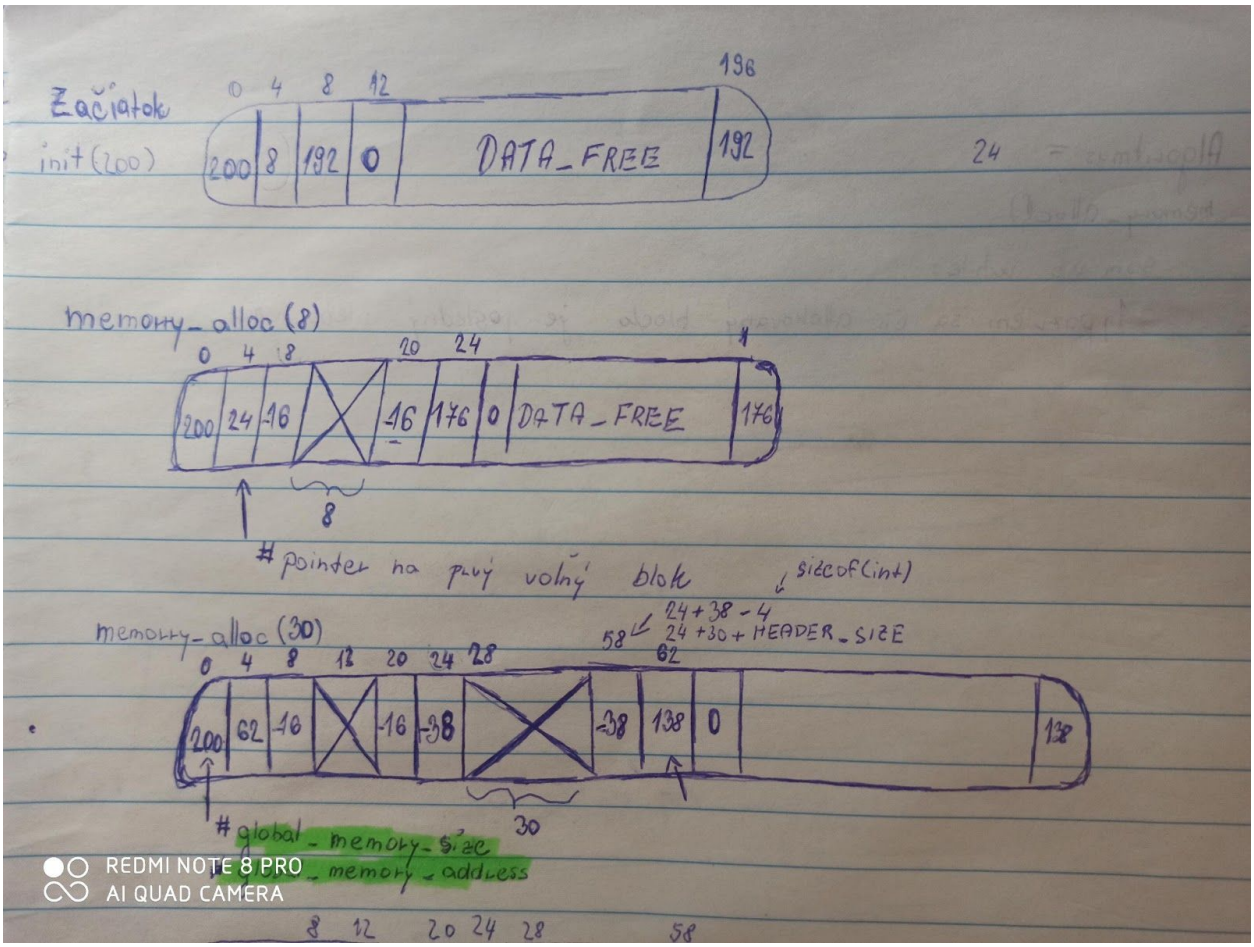
V celom programe sú definované premenné a funkcie používané v programe (v súbore header.h)

```
#define HEADER_SIZE sizeof(unsigned int)
#define FOOTER_SIZE sizeof(unsigned int)
#define FOOTER_HEADER_SIZE (2*sizeof(unsigned int))
```

Začiatky vývoja programu boli v niekoľkých nákresoch na papier “ako by to malo byť”. Môj návrh memory initu bol, že samotný region bude vyzerať ako jednotlivý blok pamäte – V prvých 4B bude mať svoju veľkosť a v druhých 8B bude mať „padding“ na prvý voľný blok, resp. jeho hlavičku. Tento pointer regionu v programe nazývam „super_pointer“.

Pre lepšie pochopenie mojej schémy regionu a následne pamäte prikladám môj náčrt, podľa ktorého som konštruoval svojho správcu pamäte.

Martin Mihalovič
Streda 18:00, ID: 96969



Prvý obrázok ilustruje moju predstavu po memory_init-e s veľkosťou regionu 200. Ako som už spomínal, sám region je reprezentovaný ako block pamäte. Super pointer (8) ukazuje na padding najbližšieho voľného bloku. Ďalšie dva kroky reprezentujú alokaciu bloku 8B a 30B.

Ak je blok už alokovaný, znamienko v hlavičke a pätičke je záporné. Preto aj v náčrte 3. pamäte je prvý blok -16B, druhý blok -38B a tretí je ešte voľný preto je 138B. Upravený je aj super pointer ktorý ukazuje na hlavičku prvého voľného bloku, teda v tomto prípade s hodnotou 138B.

Alokácia pamäte

Pri alokácií pamäte som používal Best Fit algoritmus, ktorý prechádzal celý region a vybral najvhodnejší. Alebo pokiaľ nenašiel presne vhodný blok pamäte. V tomto prípade som riešil hraničné prípady, či pamäťový blok nie je prvý/ posledný/ prvý aj posledný/ alebo je v strede. Takisto som ošetril aj prípad, keď požadovaný pamäťový blok nebolo možné alokovať, pretože takej velkosti v regione nebol voľný.

```
/**
 * Ak som na poslednom bloku a nenasiel som vhodny (vacsi) blok
*/
else if (!find_next_free_memory_block(current_free_block_padding) && best_match_value == -1) {
    printf("MEMORY ALLOC ERROR --> In region is not enough memory for this size of block\n");
    return 0;
}
```

Region som prechádzal explicitným prístupom – teda išiel som len po voľných blokoch, štýlom na princíp spájaného zoznamu, avšak pomocou paddingu (offsetu), aby som nezaberal zbytočnú pamäť pointrami v pamäti.

Uvoľnenie pamäte

Pre funkciu „memory_free“ a fragmentáciu, som si vytvoril dve pomocné funkcie v úmysle modularity kódu. Tieto funkcie sa pozrú na predchádzajúci a nasledujúci blok, a keď tieto bloky sú voľné, tak sa s nimi spojí.

Logika funkcie memory_free ale aj memory_alloc spočíva, že najskôr vyriešim bloky pamäte – rozdelím, spojím, zachovám (podľa prípadu) a v zápätí riešim pointre na nesledujúce a predchádzajúce. Taktiež podľa jednotlivého prípadu, ktorý je okomentovaný v programe.

Kontrola pamäte

Funkcia slúži ako kontrola, či daný ukazovateľ je platný a či nebol ešte uvoľnený. Logika mojej funkcie memory_check spočíva v tom, že sa nespolieham na to, že daný ukazovateľ je hlavička alebo pätička, ale pripúšťam, že to môže byť pointer na dáta v nejakom bloku. Preto celý region prechádzam od začiatku, aj po obsadených aj voľných blokoch pamäte, a sledujem či daný pointer je v niektorom z daných intervalov bloku. Ak nájdem daný interval, teda blok pamäte, pozriem sa na jeho znamienko, a ak je kladné vrátim 0, lebo znamená pointer už bol uvoľnený alebo je voľný a číslo 1 vrátim ak daný pointer je platný – teda interval (blok pamäte) je obsadený a hodnota hlavičky má záporné číslo.

Testovanie

Pri testovaní som sa riadil heslom „Not tested, not implemented“. Preto som si aj celý víkend nechal na testovanie a refaktoring, kde sa mi podarilo ošetriť viacero chýb a hraničných prípadov. Pre potreby testovania som vytvoril deväť pomocných funkcií. V hlavnej funkcii „main“ je pripravená testovacia funkcia, ktorá vedieť dialóg z testujúcim, ktorý si môže navoliť rôznu kombináciu testov aj bez toho aby poznal kód. Taktiež som sa snažil pri refaktoringu čo najviac modularizovať kód, aby nevznikali redundancie v kóde.

Martin Mihalovič
Streda 18:00, ID: 96969

Technická dokumentácia

- Testovací kompilátor: gcc version 9.2.1 20200130 (Arch Linux 9.2.1+20200130-2)

- Pamäťová zložitosť: Každý blok sa skladá z 8B hlavičky a pätičky a minimálna hodnota bloku teda je 16B. Teda ak by si užívateľ alokoval pamäť po 8B, tak polovica pamäte regionu by sa na réžiu (hlavičku a pätičku). Avsak zvolil som prístup statických hlavičiek (a pätičiek) oproti dynamickým práve kvôli výkonu programu. Dynamické hlavičky by boli pomalšie vykonávané pri alokácií a uvoľňovaní blokov.

- Časová zložitosť: minimálna, testy zbiehali pod sekundu

Záver

Pri implementácií programu som využíval explicitný prístup pamäťového manažmentu s algoritmom „best fit“ a spájaným zoznamom pomocou paddingu („offsetu“), v ktorom na seba ukazujú iba voľné bloky pamäte. Kód som sa snažil čo najviac optimalizovať a sfunkčniť.

Môj postup vývoja programu si môžete pozrieť aj na git-e. Link:
https://github.com/mihino89/memory_management