Slovenská Technická Univerzita v Bratislave
Fakulta Informatiky a Informačných Technológií

Dátové Štruktúry a Algoritmy

Zadanie 1 – Správca pamäti

Marek Adamovič

Cvičenie: Pondelok 16:00 – 17:40
Ing. Peter Pištek, PhD.

2019/2020

Použité algoritmy a riešenia

Prispôsobenie premenných v závislosti od veľkosti celkovej pamäte:

V mojom zadaní som sa rozhodol pre rozdielny typ premenných v závislosti od celkovej dostupnej pamäte. Napríklad, ak je celková pamäť menšia ako 127 bajtov, tak sa v hlavičkách bude používať na vyjadrenie informácií len 1 bajt (teda char). Ak pamäť presahuje 126 bajtov, program zvolí 2 bajty (short int) a v najväčších pamätiach dokonca 4 bajty (int). Týmto sa značne zlepšia pamäťové nároky, a to hlavne pri menších pamätiach. Ako nevýhodu vnímam, že program musí často „kontrolovať“, aká veľkosť premennej sa na začiatku zvolila.

```
31 void memory_init(void *ptr, unsigned int size){
32     if(size < 127)
33         *((int *)ptr) = 1;
34     else if(size < 32766)
35         *((int *)ptr) = 2;
36     else
37         *((int *)ptr) = 4;
```

volenie veľkosti premennej (memory_init)

```
77     max_size = GLOBAL_INT_SIZE == 1 ? *(char *)(start + 1)
78     : GLOBAL_INT_SIZE == 2 ? *(short int *)(start + 1) : *(int *)(start + 1);
```

ternárny operátor pracujúci s veľkosťou používanej premennej (memory_check)

Použitie explicitného zoznamu so segmentovanou pamäťou podľa veľkosti:

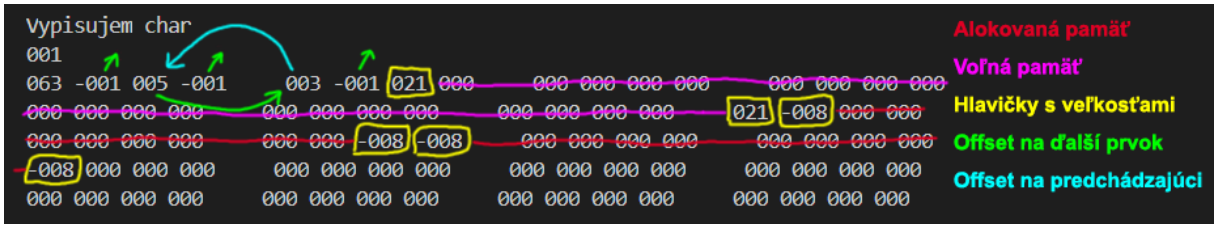
Na začiatku programu sa zvolí počet spájaných zoznamov potrebných pre chod programu v závislosti od celkovej veľkosti pamäte. Každý nasledujúci zoznam je určený pre voľné bloky 2x väčšie ako predchádzajúci, čo ušetrí pamäťové nároky pri väčších programoch. Táto metóda je značne rýchlejšia oproti klasickému lineárnemu alebo explicitnému zoznamu, keďže nemusíme prehľadávať všetky (voľné) bloky.

```
42      int max_block_size = 8, size_copy = size - 1 - GLOBAL_INT_SIZE;
43
44      while(max_block_size  < size_copy){
45          if(GLOBAL_INT_SIZE == 1)
46              *((char *)position) = -1;
47          else if(GLOBAL_INT_SIZE == 2)
48              *((short int *)position) = -1;
49          else
50              *((int *)position) = -1;
51          position += GLOBAL_INT_SIZE;
52          max_block_size *= 2;
53          size_copy -= GLOBAL_INT_SIZE;
54      }
```

vytváranie začiatkov zoznamov podľa zadanej celkovej veľkosti (memory_init)

Označovanie už alokovaných blokov pomocou znamienkového bitu:

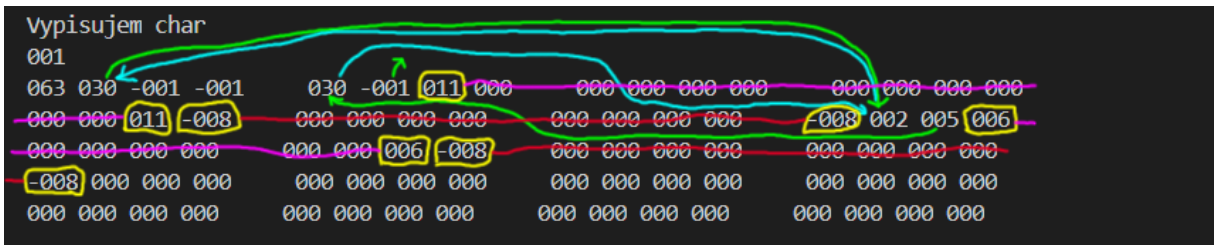
Na označenie už alokovaného bloku som si zvolil znamienkový bit v hlavičkách s veľkosťami, čo ušetrí pamäťové nároky, hlavne pri menších pamätiach. Kladná hlavička predstavuje voľný blok, zatiaľ čo záporná predstavuje už alokovaný.



organizácia pamäte pomocou spájaných zoznamov (výpis testovacej funkcie)

Používanie offsetov namiesto ukazovateľov

Program využíva výhradne offsety, čím sa zmenšia pamäťové nároky pri menších pamätiach, kde stačí offset veľkosti 1 bajt namiesto 4 bajtového ukazovateľa. Ak pripočítame k začiatku pamäte daný offset, dostaneme sa na miesto v pamäti, na ktoré „ukazuje“.



využitie ofsetov (výpis testovacej funkcie)

Absencia offsetov pri alokovaných blokoch:

Alokované bloky nepotrebujú na nič ukazovať a taktiež nič neukazuje na ne. Z toho dôvodu nemusia mať vyhradenú pamäť pre offset, čím sa znovu znížia pamäťové nároky programu.

Spájanie blokov pri uvoľňovaní pamäte:

Pri uvoľňovaní daného bloku program zisťuje, či sa vedľa neho nenachádzajú iné voľné bloky, lebo je lepšie mať jeden veľký prázdny blok ako viacero menších vedľa seba. Týmto sa zníži nežiadúca fragmentácia pamäte.

```
457
458     return merge(valid_ptr, left_check(valid_ptr), right_check(valid_ptr), -size);
459 }
```

najskôr program skontroluje, či je voľný blok vľavo, potom vpravo a s týmito informáciami spája bloky do jedného (memory_free)

Testovanie

Úvod:

Testovanie som realizoval z veľkej časti pomocou vlastnej testovacej funkcie vypisujúcej aktuálny stav pamäte. Táto testovacia funkcia prispôsobuje výpis typu premenných použitom v programe (výpisy sa mierne líšia v závislosti od veľkosti celkovej pamäte). Testovacie scenáre som organizoval tak, že najskôr som alokoval čo najväčší možný počet blokov danej veľkosti a vypočítal % efektívnosti (koľko z celkovej pamäte môžeme poskytnúť užívateľovi v danej veľkosti blokov). Potom som pristúpil ku každému alokovanému bajtu, prepísal jeho hodnotu, nechal si ju vypísať a následne uvoľnil celú pamäť. Nakoniec som znovu alokoval tým istým spôsobom tie isté bloky, porovnal či sedí efektívnosť, prepísal hodnoty alokovaných bajtov a nechal si ich vypísať.

Testovací scenár číslo 1.1:

Prideľovanie rovnakých blokov malej veľkosti (veľkosti 8 bajtov) pri použití malých celkových blokov pre správcu pamäte (50 bajtov).

```
620     //****************TESTOVACÍ SCENÁR ČÍSLO 1.1****************//
621
622     pociatocna_pamat = 50;
623     velkost_bloku = 8;
624     memory_init(memory, pociatocna_pamat);
625
626     uspesne_alokovana_1 = test_static(velkost_bloku);
627     uspesne_alokovana_2 = test_static(velkost_bloku);
628
```

```
Vypisujem char
001
063 -001 -001 -001     -014 008 007 006     005 004 003 002     001 000 000 000
000 000 000 -014     -008 008 007 006     005 004 003 002     001 -008 -008 008
007 006 005 004     003 002 001 -008     -008 008 007 006     005 004 003 002
001 -008 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000

Pociatocna pamat: 50
Velkost alokovanych blokov: 8
Uspesne alokovana: 32
Efektivnost: 64.00%
Efektivnosti sa rovnaju
```

Testovací scenár číslo 1.2:

Prideľovanie rovnakých blokov malej veľkosti (veľkosti 24 bajtov) pri použití malých celkových blokov pre správcu pamäte (200 bajtov [+1 pre zrozumiteľnejší výpis testovacej funkcie]).

```
622     //****************TESTOVACÍ SCENÁR ČÍSLO 1.2****************//
623
624     pociatocna_pamat = 201;
625     velkost_bloku = 24;
626     memory_init(memory, pociatocna_pamat);
627
628     uspesne_alokovana_1 = test_static(velkost_bloku);
629     uspesne_alokovana_2 = test_static(velkost_bloku);
630
```

```
Vypisujem short
00002
00255 -00001 00013 -00001     -00001 -00001 00005 -00001     00014 00000 00000 00000
00000 00000 00000 00000     00014 -00024 00024 00022     00020 00018 00016 00014
00012 00010 00008 00006     00004 00002 -00024 -00024     00024 00022 00020 00018
00016 00014 00012 00010     00008 00006 00004 00002     -00024 -00024 00024 00022
00020 00018 00016 00014     00012 00010 00008 00006     00004 00002 -00024 -00024

Pociatocna pamat: 201
Velkost alokovanych blokov: 24
Uspesne alokovana: 144
Efektivnost: 71.64%
Efektivnosti sa rovnaju
```

Testovací scenár číslo 2.1:

Prideľovanie náhodných blokov malej veľkosti (veľkosti 8 - 24 bajtov) pri použití malých celkových blokov pre správcu pamäte (50 bajtov).

```
644     //****************TESTOVACÍ SCENÁR ČÍSLO 2.1****************//
645
646     pociatocna_pamat = 50;
647     velkost_bloku_od = 8;
648     velkost_bloku_do = 24;
649     memory_init(memory, pociatocna_pamat);
650
651     uspesne_alokovana_1 = test_random(velkost_bloku_od, velkost_bloku_do);
652     uspesne_alokovana_2 = test_random(velkost_bloku_od, velkost_bloku_do);
```

```
Vypisujem char
001
063 005 -001 -001     002 -001 014 000     000 000 000 000     000 000 000 000
000 000 000 000     000 014 -016 016     015 014 013 012     011 010 009 008
007 006 005 004     003 002 001 -016     -008 008 007 006     005 004 003 002
001 -008 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000
000 000 000 000     000 000 000 000     000 000 000 000     000 000 000 000

Pociatocna pamat: 50
Velkost alokovanych blokov: 8 - 24
Uspesne alokovana prva: 24
Uspesne alokovana druha: 24
Efektivnost prva: 48.00%
Efektivnost druha: 48.00%
```

Testovací scenár číslo 2.2:

Prideľovanie náhodných blokov malej veľkosti (veľkosti 8 - 24 bajtov[pre krajší výpis funkcie budeme v tomto scenári uvažovať len párne veľkosti]) pri použití malých celkových blokov pre správcu pamäte (200 bajtov [+1 pre zrozumiteľnejší výpis testovacej funkcie]).

```
644     //****************TESTOVACÍ SCENÁR ČÍSLO 2.2****************//
645
646     pociatocna_pamat = 201;
647     velkost_bloku_od = 8;
648     velkost_bloku_do = 24;
649     memory_init(memory, pociatocna_pamat);
650
651     uspesne_alokovana_1 = test_random(velkost_bloku_od, velkost_bloku_do);
652     uspesne_alokovana_2 = test_random(velkost_bloku_od, velkost_bloku_do);
```

```
Vypisujem short
00002
00255 -00001 -00001 -00001     -00001 -00001 -00026 00016     00014 00012 00010 00008
00006 00004 00002 00000     00000 00000 -00022 00011     -00026 -00008 00008 00006
00004 00002 -00008 -00008     00008 00006 00004 00002     -00008 -00024 00024 00022
00020 00018 00016 00014     00012 00010 00008 00006     00004 00002 -00024 -00016
00016 00014 00012 00010     00008 00006 00004 00002     -00016 -00008 00008 00006
00004 00002 -00008 -00008     00008 00006 00004 00002     -00008 -00024 00024 00022
00020 00018 00016 00014     00012 00010 00008 00006     00004 00002 -00024 -00020
00020 00018 00016 00014     00012 00010 00008 00006     00004 00002 -00020 -00008
00008 00006 00004 00002     -00008 00000 00000 00000     00000 00000 00000 00000
00000 00000 00000 00000     00000 00000 00000 00000     00000 00000 00000 00000

Pociatocna pamat: 201
Velkost alokovanych blokov: 8 - 24
Uspesne alokovana prva: 144
Uspesne alokovana druha: 140
Efektivnost prva: 71.64%
Efektivnost druha: 69.65%
```

Testovací scenár číslo 3.1:

Prideľovanie náhodných blokov väčšej veľkosti (veľkosti 500 - 5000 bajtov) pri použití väčších celkových blokov pre správcu pamäte (50 000 bajtov).

```
661     //****************TESTOVACÍ SCENÁR ČÍSLO 3.1****************//
662
663     pociatocna_pamat = 50000;
664     velkost_bloku_od = 500;
665     velkost_bloku_do = 5000;
666     memory_init(memory, pociatocna_pamat);
667
668     uspesne_alokovana_1 = test_random(velkost_bloku_od, velkost_bloku_do);
669     uspesne_alokovana_2 = test_random(velkost_bloku_od, velkost_bloku_do);
```

```
Pociatocna pamat: 50000
Velkost alokovanych blokov: 500 - 5000
Uspesne alokovana prva: 49680
Uspesne alokovana druha: 49652
Efektivnost prva: 99.36%
Efektivnost druha: 99.30%
```

efektívnosť môže kolísať podľa poslednej vygenerovanej veľkosti (program neskúša šťastie, ak je požiadavka privysoká)

Testovací scenár číslo 4.1:

Prideľovanie náhodných blokov väčšej veľkosti (veľkosti 8 – 50 000 bajtov) pri použití väčších celkových blokov pre správcu pamäte (500 000 bajtov).

```
678     //***************TESTOVACÍ SCENÁR ČÍSLO 4.1***************//
679
680     pociatocna_pamat = 500000;
681     velkost_bloku_od = 8;
682     velkost_bloku_do = 50000;
683     memory_init(memory, pociatocna_pamat);
684
685     uspesne_alokovana_1 = test_random(velkost_bloku_od, velkost_bloku_do);
686     uspesne_alokovana_2 = test_random(velkost_bloku_od, velkost_bloku_do);
```

```
Pociatocna pamat: 500000
Velkost alokovanych blokov: 8 - 50000
Uspesne alokovana prva: 493696
Uspesne alokovana druha: 482600
Efektivnost prva: 98.74%
Efektivnost druha: 96.52%
```

Záver

Časová zložitosť program je pri najhoršom prípade (ten, kde musíme prechádzať všetky bloky v zozname, ktoré by mohli sedieť veľkosťou) je O(n+1), kde n je počet blokov v danom veľkostnom zozname. Akonáhle prejdeme na väčší zoznam, tak prvý voľný blok môžeme použiť, keďže jeho veľkosť bude určite väčšia ako veľkosť, ktorú požaduje užívateľ.

Čo sa týka pamäťovej zložitosti, tak tú môžeme vyjadriť % úspešne alokovaných bajtov oproti počtu bajtov v pamäti (efektivita). Pri menších pamätiach (50 - 200 bajtov) sa efektivita pohybuje v priemere okolo 60%. Tu môže značne kolísať v závislosti od veľkosti požadovaných blokov (ak užívateľ požaduje 24 bajtové bloky, tak v 50 bajtovej pamäti by sme len veľmi ťažko pomestili 2 bloky aj s réžiou pamäte). Pri väčších pamätiach (1000+ bajtov) je efektivita značne lepšia, keďže nároky na réžiu pamäte ostávajú skoro nezmenené. Tu hrajú rolu veľkosti požadovaných blokov, keďže každý blok používa veľkostnú hlavičku a pätičku. Pri väčších blokoch dosahuje efektivita hodnotu až 98+%, pri menších požadovaných blokoch je to o pár % menej.