Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dátové štruktúry a algoritmy

Zadanie 1 – správa pamäti

LS 2020/2021

Samuel Kováč

Študijný program: B-INFO4
Vyučujúci: Ing. Lukáš Kohútka, PhD.
Cvičenia: Pondelok 9:00
Marec 2021

Samuel Kováč
DSA Zadanie 1

Vypracovanie zadania

Hlavička v mojom riešení sa skladá z 4 bytov (integer), pričom voľnosť bloku pamäte určujem podľa znamienka čísla v hlavičke. Ak je dané číslo kladné, znamená to že je blok voľný a naopak. Pri inicializácií si tiež nastavím pätu pamäte na nulu (funguje ako ukončovací znak). Program využíva len jednu globálnu premennú ***m*** ktorá slúži ako ukazovateľ na začiatok pamäte. Algoritmus alokovania pamäte ktorý som použil, sa nazýva First fit. Jeho výhodou je rýchlosť, keďže alokuje hneď prvý voľný blok pamäte a nerobí ďalšie zložité porovnávania. Nevýhodou je ale zvýšená externá fragmentácia oproti algoritmu Best fit. First fit tiež obvykle zanecháva väčší voľný blok na konci pamäte. Na základe tohto algoritmu vieme určiť časovú [O(n)] a pamäťovú [O(n)] náročnosť programu.

memory_alloc

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

```
void *memory_alloc(size_t size){
    void *ptr = m;
    while(((*((int*)ptr) < size) || (!(is_free(*((int*)ptr))))) && (*((char*)ptr) != 0))
        ptr += abs(*((int*)ptr)) + sizeof(int);
```

Na začiatku funkcie pomocou while loopu nájdeme prvý voľný blok pamäte.

Prvá podmienka kontroluje či sa veľkosť nájdeného bloku rovná požadovanej veľkosti. Ak áno, nastaví znamienko hlavičky na (-) a funkcia vráti pointer na alokovaný blok, posunutý o veľkosť hlavičky.

```
if((*((int*)ptr)) == size) {
    *((int*)ptr) = *((int*)ptr) * (-1);
    return (void*)(ptr + sizeof(int));
}
```

V prípade že veľkosť voľného bloku je väčšia ako požadovaná veľkosť, voľný blok je treba rozdeliť. Najprv skontroluje či po rozdelení bloku nezostane voľný blok veľmi malej veľkosti (5B – 4B hlavička) - v prípade že zvyšný blok je veľmi malý, voľný blok nebude rozdeľovať. Ak je zvyšok dostatočne veľký, funkcia voľný blok rozdelí a vráti pointer na alokovaný blok, posunutý o veľkosť hlavičky.

```
else if((*((int*)ptr)) > size) {

    void *z = ptr + sizeof(int) + size;
    int r = (*((int*)ptr)-size-sizeof(int));
    if (r < 1) {
        *((int*)ptr) = *((int*)ptr) * (-1);
        return (void*)(ptr + sizeof(int));
    }
    *((int*)z) = *((int*)ptr)-size-sizeof(int);
    *((int*)ptr)=size;
    *((int*)ptr)= *((int*)ptr) * (-1);
    return (void*)(ptr + sizeof(int));
}
```

Nakoniec ak sa blok do teraz nepodarilo alokovať, znamená to že neexistuje voľný blok s dostatočnou veľkosťou a funkcia vráti NULL.

```
else {
    printf("Nedostatok miesta pre velkost %d\n", size);
    return NULL;
}
```

Samuel Kováč
DSA Zadanie 1

memory_init()

Funkcia memory_init slúži na inicializáciu spravovanej voľnej pamäte. Predpokladajte, že funkcia sa volá práve raz pred všetkými inými volaniami memory_alloc, memory_free a memory_check. Viď testovanie nižšie. Ako vstupný parameter funkcie príde blok pamäte, ktorú môžete použiť pre organizovanie a aj pridelenie voľnej pamäte. Vaše funkcie nemôžu používať globálne premenné okrem jednej globálnej premennej na zapamätanie ukazovateľa na pamäť, ktorá vstupuje do funkcie memory_init. Ukazovatele, ktoré prideľuje vaša funkcia memory_alloc musia byť výhradne z bloku pamäte, ktorá bola pridelená funkcii memory_init.

Funkcia memory_init najskôr nastaví celé pole pamäte na 1tky, potom nastaví globálny pointer (m) na začiatok pamäte. Nakoniec nastavíme do hlavičky veľkosť pamäte (odčítame veľkosť hlavičky a päty) a nastavíme pätu na 0 ako koncový znak poľa.

```
void memory_init(void *ptr, int size) {
    for(int i = 0; i < size; i++)
        *((char*)ptr + i) = 1;
    m=ptr;
    *((int*)ptr) = size - sizeof(int) - sizeof(int);
    *((int*)(ptr+size-sizeof(int))) = 0;
}
```

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. Môžete predpokladať, že parameter bude vždy platný ukazovateľ, vrátený z predchádzajúcich volaní funkcie memory_alloc, ktorý ešte nebol uvoľnený.

Funkcia memory_free najprv odčíta veľkosť hlavičky od zadaného pointera, následne skontroluje pointer funkciou memory_check a uvoľní ho pomocou otočenia znamienka hlavičky na kladnú hodnotu.

```
int memory_free(void *valid_ptr) {
    if(valid_ptr)
        valid_ptr -= sizeof(int);
    if(memory_check(valid_ptr)) {
        *((int*)valid_ptr) = *((int*)valid_ptr) * (-1);
        memory_merge();
        return 1;
    }
    printf("Neplatny pointer, nemozno uvolnit\n");
    return 0;
}
```

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 memory_check skontroluje či sa pointer nerovná NULL alebo nie je voľný a podľa toho vráti 0/1

```
int memory_check(void *ptr) {
    if (!ptr || is_free(*((int*)ptr)))
        return 0;
    else
        return 1;
}
```

Samuel Kováč
DSA Zadanie 1

memory_merge()

```
void memory_merge() {
    void *ptr = m;
    while(*((char*)ptr) != 0) {
        if(is_free(*((int*)ptr)) && is_free(*((int*)(ptr + abs(*((int*)ptr)) + sizeof(int)))) && (*((char*)(ptr + abs(*((int*)ptr)) + sizeof(int))) != 0)) {
            *((int*)ptr) += (*((int*)(ptr + *((int*)ptr) + sizeof(int))) + sizeof(int));
        }
        ptr += abs(*((int*)ptr)) + sizeof(int);
    }
}
```

Pomocná funkcia ktorej úlohou je pospájať susediace voľné bloky. Ak sú za sebou 2 voľné bloky, k veľkosti prvého bloku pripočíta veľkosť druhého + veľkosť hlavičky druhého.

vypis()

Zabezpečuje vizuálne zobrazenie rozloženia pamäte. Zobrazí všetky bloky pamäte ako číslo predstavujúce veľkosť daného bloku (bez hlavičky). Ak je číslo kladné, daný blok je voľný a naopak.

```
void vypis() {
    void *ptr = m;
    printf("| ");
    while(*((char*)ptr) != 0) {
        printf("%d | ", *((int*)ptr));
        ptr += abs(*((int*)ptr)) + sizeof(int);
    }
    printf("\n");
}
```

Testy

1. test

Prideľovanie rovnakých blokov malej veľkosti (12 bytov) pri použití malých celkových blokov pre správcu pamäte (100bytov)

```
Test1 :
| 92 |
| -12 | 76 |
| -12 | -12 | 60 |
| -12 | -12 | -12 | 44 |
| -12 | -12 | -12 | -12 | 28 |
| -12 | -12 | -12 | -12 | -12 | 12 |
| -12 | -12 | -12 | -12 | -12 | -12 |
Nedostatok miesta
| -12 | -12 | -12 | -12 | -12 | -12 |
Nedostatok miesta
| -12 | -12 | -12 | -12 | -12 | -12 |
| 12 | -12 | -12 | -12 | -12 | -12 |
| 28 | -12 | -12 | -12 | -12 |
| 44 | -12 | -12 | -12 |
| 60 | -12 | -12 |
| 76 | -12 |
| 92 |
Neplatny pointer, nemozno uvolnit
Neplatny pointer, nemozno uvolnit
Test1 prebehol uspesne
```

Priebeh pamäte počas testu

Úspešnosť alokovania oproti ideálnemu riešeniu : ***62.5%***

Samuel Kováč
DSA Zadanie 1

2. test

Prideľovanie nerovnakých blokov malej veľkosti (náhodné veľkosti 8 až 24bytov)pri použití malých celkových blokov pre správcu pamäte (100 bytov)

```
Test2 :
| 92 |
| -15 | 73 |
| -15 | -13 | 56 |
| -15 | -13 | -18 | 34 |
| -15 | -13 | -18 | -22 | 8 |
Nedostatok miesta
| -15 | -13 | -18 | -22 | 8 |
Nedostatok miesta
| -15 | -13 | -18 | -22 | 8 |
Nedostatok miesta
| -15 | -13 | -18 | -22 | 8 |
Nedostatok miesta
| -15 | -13 | -18 | -22 | 8 |
| 15 | -13 | -18 | -22 | 8 |
| 32 | -18 | -22 | 8 |
| 54 | -22 | 8 |
| 80 | 8 |
Neplatny pointer, nemozno uvolnit
Neplatny pointer, nemozno uvolnit
Neplatny pointer, nemozno uvolnit
Neplatny pointer, nemozno uvolnit
Test2 prebehol uspesne
```

Priebeh pamäte počas testu

*Úspešnosť alokovania oproti ideálnemu riešeniu : **65 - 75%***

3. test

Prideľovanie nerovnakých blokov väčšej veľkosti (veľkosti 500 až 5000 bytov) pri použití väčších celkových blokov pre správcu pamäte (10 000 bytov)

```
Test3 :
| 9992 |
| -2858 | 7130 |
| -2858 | -1841 | 5285 |
| -2858 | -1841 | -2087 | 3194 |
Nedostatok miesta pre velkost 4385
| -2858 | -1841 | -2087 | 3194 |
| -2858 | -1841 | -2087 | -1275 | 1915 |
| -2858 | -1841 | -2087 | -1275 | -1797 | 114 |
Nedostatok miesta pre velkost 2466
| -2858 | -1841 | -2087 | -1275 | -1797 | 114 |
Nedostatok miesta pre velkost 4047
| -2858 | -1841 | -2087 | -1275 | -1797 | 114 |
| 2858 | -1841 | -2087 | -1275 | -1797 | 114 |
| 4703 | -2087 | -1275 | -1797 | 114 |
| 6794 | -1275 | -1797 | 114 |
Neplatny pointer, nemozno uvolnit
| 8073 | -1797 | 114 |
| 9874 | 114 |
Neplatny pointer, nemozno uvolnit
Neplatny pointer, nemozno uvolnit
Test3 prebehol uspesne
```

Priebeh pamäte počas testu

*Úspešnosť alokovania oproti ideálnemu riešeniu : **100%***

Samuel Kováč
DSA Zadanie 1

4. test

Prideľovanie nerovnakých blokov malých a veľkých veľkostí (veľkosti od 8 bytov do 50 000 bytov) pri použití väčších celkových blokov pre správcu pamäte (100 000 bytov).

```
Test4 :
| 99992 |
| -17852 | 82136 |
| -17852 | -19365 | 62767 |
| -17852 | -19365 | -15167 | 47596 |
| -17852 | -19365 | -15167 | -15839 | 31753 |
| -17852 | -19365 | -15167 | -15839 | -361 | 31388 |
| -17852 | -19365 | -15167 | -15839 | -361 | -21881 | 9503 |
Nedostatok miesta pre velkost 14960
| -17852 | -19365 | -15167 | -15839 | -361 | -21881 | 9503 |
Nedostatok miesta pre velkost 28804
| -17852 | -19365 | -15167 | -15839 | -361 | -21881 | 9503 |
| 17852 | -19365 | -15167 | -15839 | -361 | -21881 | 9503 |
| 37221 | -15167 | -15839 | -361 | -21881 | 9503 |
| 52392 | -15839 | -361 | -21881 | 9503 |
| 68235 | -361 | -21881 | 9503 |
| 68600 | -21881 | 9503 |
| 90485 | 9503 |
Neplatny pointer, nemozno uvolnit
Neplatny pointer, nemozno uvolnit
Test4 prebehol uspesne
```

Priebeh pamäte počas testu

*Úspešnosť alokovania oproti ideálnemu riešeniu : **99.7 - 99.94%***

Záver testovania

Z testovania vyplýva že pamäť pridelená funkciou memory_alloc nepresahuje pôvodný blok ani neprekrýva doteraz pridelenú pamäť, a dá sa úspešne uvoľniť funkciou memory_free.