Slovenská technická univerzita, Fakulta informatiky a informačných technológií

Dátové štruktúry a algoritmy

Zadanie č.1 – Správca pamäti

Jana Vrabľová
2.ročník
2016/2017

1. časť -Testovač prideľovania pamäte

Testovač prideľovania pamäte v 10 cykloch opakuje napĺňanie a uvoľnovanie pamäte.
Testovač alokuje súvisly blok vo velkosti 50 B a následne testuje alokáciu polí v 4 scenároch:

- prideľuje náhodné bloky vo veľkostiach 8 až 24 B
- prideľuje rovnaké bloky náhodnej veľkosti od 8 až 24 B
- prideľuje náhodné bloky od 500 do 5000 B
- prideľuje náhodné bloky od 8 do 50 000B

Testovanie teda prebieha ako vytváranie dvojrozmerných polí. Každé pole je naplnené hodnotou prvého indexu dvojrozmerného poľa:

```
pam_bloky[i]=(uint8_t*)memory_alloc (velkosti[i] * sizeof(uint8_t));
   for (j=0; j<velkosti[i]; j++){
     pam_bloky[i][j]=i;
   }
```

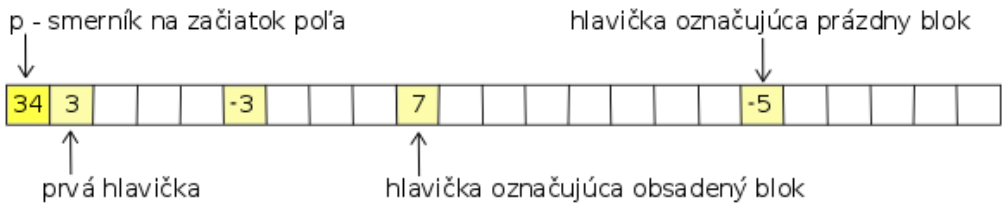
Premenná *n_hodnota* je nastavená na 0. Pri uvoľnovaní pamäte sa kontrolú bloky, či sú správne naplnené. Pokiaľ nie, návratová hodnota sa nastaví na 1. Ak testy prebehli v poriadku funkcia vráti 0 ak nie vráti 1.

Uvoľnovanie pamäte a kontrola, či sú bloky správne naplnené:

```
for(i=0; i<poc_byte; i++){
     for(j=0; j<velkosti[i]; j++){
        if(pam_bloky[i][j]!=i){
           n_hodnota=1;
         }
                  }
     memory_free(pam_bloky[i]);
         }
  memory_free(pam_bloky);
  memory_free(velkosti);
```

2. časť – Vlastný algoritmus pridelovania pamäti

Spôsob prideľovania pamäte:



Princíp funkcie spočíva v tom, že na začiatku sa vytvorí veľké pole z ktorej funkcia memory_alloc bude prideľovať menšie pamäťové bloky. Každé pole bude mať hlavičku o veľkosti 1 int, čo je vhodné z hľadiska pamäťovej náročnosti. Hodnota uložená v hlavičke predstavuje veľkosť bloku ktorý je možné alokovať ak nie je obsadený, prípadne veľkosť už obsadeného bloku. Pokiaľ je veľkosť záporná blok je neobsadený a ak spĺňa podmienky funkcia vráti smerník na toto pole. Vo

funkcii je použitý algoritmus first fit.
Funkcia memory_init inicializuje globálnu premennú p, čo je smerník ukazujúci na začiatok poľa. Do poľa sa na pozíciu s indexom 0 priradí veľkosť celého poľa (zmenšená o veľkosť pamäte kde je táto hodnota uložená). Na nasledujúce miesto sa vytvorí prvá hlavička do ktorej sa zapíše záporná hodnota veľkosti poľa, čo znamená, že blok je neobsadený.

Funkcia memory_alloc - vytvoria sa 2 smerníky. Smerníkom *hlavicka* sa bude prechádzať celé pole a smerník *dalsia* sa používa v prípade, že nájdený blok je väčší ako je potrebné. Zvyšná pamäť je uvoľnená tak, že sa vytvorí ďalšia hlavička. Smerník *hlavicka* sa nastaví na prvú hlavičku a v cykle sa prechádza všetkými hlavičkami, pričom sa preskakujú všetky obsadené alebo také, ktoré majú malú veľkosť. Keď sa nájde prvý vhodný blok veľkosť sa prepíše na tú, ktorú chceme alokovať. Smerník *dalsia* sa posunie za nové pole a vytvorí ďalšiu hlavičku, ktorá bude neobsadená. Uvoľnením nepotrebnej časti pamäte sa zamedzuje vnútornej fragmentácií pamäte. Funkcia vráti smerník posunutý za *hlavicku* ukazujúci na začiatok prideleného poľa. Pokiaľ nenájde žiadny vhodný blok pamäte vráti NULL.

Alokácia pamäte:
```
if (size >= abs(*hlavicka) - sizeof(int)) {
                (*hlavicka)= abs(*hlavicka);
        } else {
                zvysok = abs(*hlavicka)-size-sizeof(int);
                (*hlavicka)= size;
                dalsia = (int*)(((char*)hlavicka) + sizeof(int) + (*hlavicka));
                (*dalsia) = -zvysok;
        }
        return hlavicka+1;
```

Najhorší možný prípad časovej zložitosti tohto riešenia je O(n), pamäťovej zložitosti tiež O(n).

Funkcia memory_check zisťuje, či je smerník platný. Ak nie je platný vráti 0, ak je platný vráti 1. Pomocný smerník *platný* je na začiatku nastavený na prvú hlavičku a v cykle prechádza po všetkých hlavičkách pokiaľ je adresa pomocného smerníka *platný* menšia ako adresa smerníka, ktorý hľadáme (ten musí byť posunutý o 1 int čo je veľkosť hlavičky, aby ukazovali na rovnakú adresu). Ak už adresa nie je menšia mal by smerník ukazovať na tú istú adresu ako smerník *platný.* Ak sa adresy nezhodujú smerník nie je platný a ukazuje do vnútra nejakého bloku. Tiež ak smerník ukazuje na NULL alebo na uvoľnený blok je tiež neplatný a memory_check vráti 0.

Zisťovanie, či smerník neukazuje do vnútra bloku:
```
for (platny =((int*)p) + 1; platny < (int*)ptr - 1; platny =(int*)(((char*)platny) + abs(*platny) +
sizeof(int)))
        {
                continue;
        }

        if(platny != ((int*)ptr) - 1){
                return 0;
        }
```
Najhorší možný prípad časovej zložitosti tohto riešenia je O(n), pamäťovej zložitosti je O(n).

Funkcia memory_free pomocou while cyklu vyhľadá blok pamäte, ktorý chceme uvoľniť. Smerník *predchadzajuci* je nastavený na NULL ak by sme uvoľňovali prvý blok. Inak sa vo while cykle nastavuje na predchadzajúcu hlavičku. Ak predchádzajúci blok existuje (uvolňovaný blok nie

je prvý) a súčasne je predchádzajúci neobsadený, oba bloky spojíme, nová hlavička je hlavička predchadzajúceho a veľkosti oboch blokov sa zrátajú.

Spájanie s predchádzajucim blokom:

```
 if (predchadzajuci!=NULL && (*(int*)predchadzajuci) < 0){  /*spojenie s predchadzajucim volnym blokom*/
     (*(int*)predchadzajuci) = abs(*(int*)predchadzajuci);
          (*(int*)predchadzajuci) += ((*(int*)ptr) + sizeof(int));
          ptr = predchadzajuci;
  }
```

Ak nie sme na konci poľa a nasledujúci blok je voľný do veľkosti hlavičky pripočítame veľkosť nasledujúceho bloku, čím ich spojíme. Hodnotu zmeníme na zápornu. Blok je uvoľnený pre ďalšiu alokáciu. Spájanie voľných blokov zamedzuje vonkajšej fragmentácii pamäte.

Spájanie s nasledujúcim blokom:

```
if((dalsi) - p < velkost && (*(int *)dalsi) < 0){
          (*(int *)dalsi) = abs(*(int *)dalsi);
          (*(int*)ptr) += (*(int *)dalsi) + sizeof(int);
  }

  (*(int*)ptr) = -(*(int*)ptr);
```

Časová zložitosť funkcie je O(n), pamäťová zložitosť je O(n).