Samuel Bubán 9.3.2020

DSA Zadanie 1

Analýza:

- na vyriešenie úlohy som použil niekoľko spájaných zoznamov s voľnými blokmi pamäti, počet zoznamov je závislý od veľkosti pamäte
- pre zapisovanie hodnôt som použil dynamické hlavičky – ak je veľkosť dostatočná, aby sa čísla zapísali len do charov, tak týmto ušetrím niekoľko bajtov pamäte
    - pri malých veľkostiach namiesto 8-bajtových hlavičiek s pätou mám len 2-bajtové
    - nemám presne určené, koľko bajtov obsahujú moje premenné, preto sa najprv overí, aká maximálna hodnota sa do nich zmestí, a až podľa toho je určené, ktorý typ sa v programe využije
- program je optimalizovaný pre pamäťovú zložitosť, a teda využíva pre každý blok len hlavičku a pätu o veľkosti typu – podľa celej pamäte, a najmenší možný alokovateľný blok je o veľkosti 2x veľkosť typu, ktorý sa používa, ak si ale používateľ vypýta menšiu veľkosť, bude automaticky zmenená na minimálnu
- časová zložitosť je optimalizovaná vrámci možností, ale tiež je najviac efektívny ako môže byť
- svoje riešenie som rozšíril, takže dokáže podporovať bloky aj pamäť o veľkosti 8 exabajtov (8 000 petabajtov, 8 000 000 terabajtov)
- výsledný projekt som testoval na zadaných parametroch (pre malé a veľké pamäte, pre malé a veľké bloky), ale takisto som svoj program otestoval na projekte, ktorý som robil minulý semester na PRPR – zoznam, ktorý využíva prideľovanie a uvoľňovanie pamäte, a tento test prešiel bezproblémovo
- vo svojom projekte som použil makrá na čítanie a zápis do pamäte, a takto som si uľahčil celú prácu, následne som pridal dynamické hlavičky, a musel som pre čítanie a zápis vytvoriť funkcie, ktoré sa rozhodovali, aký typ budú z pamäte čítať alebo zapisovať

memory_init:

- na začiatku sa nastaví prvý bajt pamäte na index, ktorý je určený podľa veľkosti pamäte, a určuje, aká veľkosť hlavičky sa bude používať (0 – char, 1 – short, 2 – int, 3 – long long)
- následne sa nastavia pointery na začiatku pamäte na nulu, a index (prvý bajt pamäte) sa nastaví na štvornásobok počtu pointerov + index, takto bude obsahovať aj počet pointerov aj typ hlavičky, čo sa používa
- potom sa už len inicializuje prvý blok pamäte (nastavia sa hranice na jeho veľkosť (setBoundaries()), a nastavia sa pointery vnútri na nasledujúci prvok (prázny) a predchádzajúci (jeden z pointerov na začiatku, posledný, teda najväčší)), a pointer v zozname s najväčšími pointermi sa nastaví na jediný existujúci blok

Samuel Bubán 9.3.2020

memory_alloc:

- na začiatok sa zavolá funkcia pre nájdenie voľného bloku v pamäti (memFindFree()) – tá na začiatok určí podľa veľkosti, ktorý zoznam by mal obsahovať ideálnu voľnú pamäť, potom nájde najmenší vhodný blok, a vráti ho naspäť do memory_alloc
    - ak sa v tomto zozname dostatočný blok nenachádza, prejde aj nasledujúci zoznam blokov, ktoré sú zaručené, že budú mať dostatočnú veľkosť
    - celý zoznam potrebujem raz prejsť (ak sa nenájde ideálny blok – presná zhoda veľkosti), a potom dokážem vrátiť najmenší pasujúci blok
- tento blok sa vymaže zo zoznamu voľných blokov (eraseFromList()), a zmení sa na obsadený (prepíše sa znamienko určujúce jeho veľkosť na záporné)
- ak je v tom bloku dostatok miesta na ďalší, tak sa rozdelí na dva – prvý obsadený a druhý voľný, ktorý sa pridá na začiatok zoznamu (split())

memory_check:

- najprv prejde celú pamäť po blokoch (ako pri implicitných zoznamoch), a zistí, či pointer pasuje na niektorý z blokov
- ak pasuje, overí hlavičku a pätu tohto bloku, či sa rovnajú a sú záporné

memory_free:

- na začiatku zavolá **memory_check** pre overenie, či sa jedná o správny pointer (zaplnený blok v pamäti)
- na začiatku sa označí blok ako voľný, a pridá sa medzi voľné bloky (addToListStart())
- ak je blok vpravo od uvoľňovaného voľný, zavolá sa funkcia, ktorá ich oba vymaže zo zoznamu voľných blokov, spojí a pridá na začiatok (mergeRight())
- ak je blok vľavo voľný, znovu sa zavolá tá istá funkcia, a spoja sa bloky znovu

Samuel Bubán 9.3.2020

Časová zložitosť:

memory_init:

- časová zložitosť je závislá od veľkosti pamäte – väčšia pamäť potrebuje viac pointerov, je to približne logaritmus(n) so základom 4

memory_alloc:

- vyhľadanie voľnej pamäte je o(n), a závisí od fragmentácie pamäte – počet blokov podobnej veľkosti ako je hľadaný
- zvyšok je konštantná zložitosť

memory_check:

- konštantná zložitosť, nezávislá od ničoho

memory_free:

- konštantná zložitosť, nezávislá od ničoho (jemne sa mení na základe voľných blokov okolo)

Možné optimalizácie:

- zoznam voľných blokov by sa dal usporiadať, a tak urýchliť hľadanie voľného bloku, a dal by sa teda použiť first fit (lebo by to bol zároveň aj best fit), teraz musím prejsť všetky bloky v zozname podobne veľkých blokov, iba ak nájdem presnú zhodu, vtedy sa zastaví
- ak by som zmenil spôsob určovania bloku, či je plný alebo nie na párne/nepárne čísla, zvýšila by sa trochu pamäťová zložitosť (o maximálne 1B na blok pamäte), ale mohol by som podporovať dvojnásobne veľké bloky aj pamäť (16 exabajtov)

Samuel Bubán 9.3.2020