Lukáš Michalík, ID:103052
Pondelok 09:00

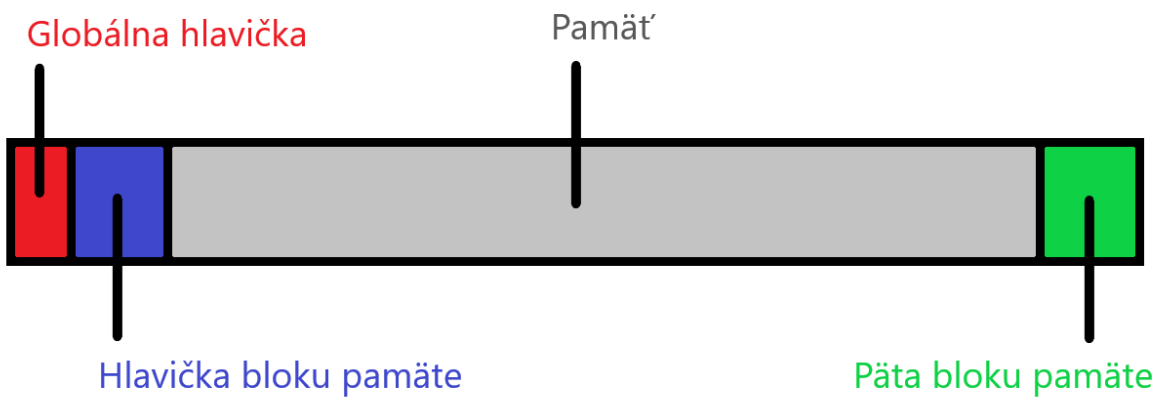
Slovenská Technická Univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dátové štruktúry a algoritmy

Zadanie 1 – Správca pamäti

Lukáš Michalík, ID:103052
Pondelok 09:00

Funkcia memory_init



Funkcia **memory_init,** ma za úlohu prvotnú inicializáciu spravovanej voľnej pamäte. Ako vstupné parametre dostane ukazovateľ na začiatok bloku pamäte a veľkosť. Následne vytvorí globálnu hlavičku danými s parametrami podľa veľkosti bloku pamäte. Ďalej vytvorí prvotnú hlavičku aj pätu a sprístupní zostávajúcu veľkosť bloku a voľnú pamäť pre užívateľa, ktorý ju môže používať.

Globálna hlavička má veľkosť 8B uchováva v sebe informáciu o veľkosti celkového bloku pamäte (4B) a celkový počet blokov (alokovaných/voľných) (4B), ktoré sa v pamäti nachádzajú.

Hlavička bloku pamäte má veľkosť 12B. Obsahuje informácie o veľkosti alokovaného/voľného bloku (4B), offset hodnotu pre najbližší nasledujúci blok (4B) a offset hodnotu pre najbližší predchádzajúci blok (4B).

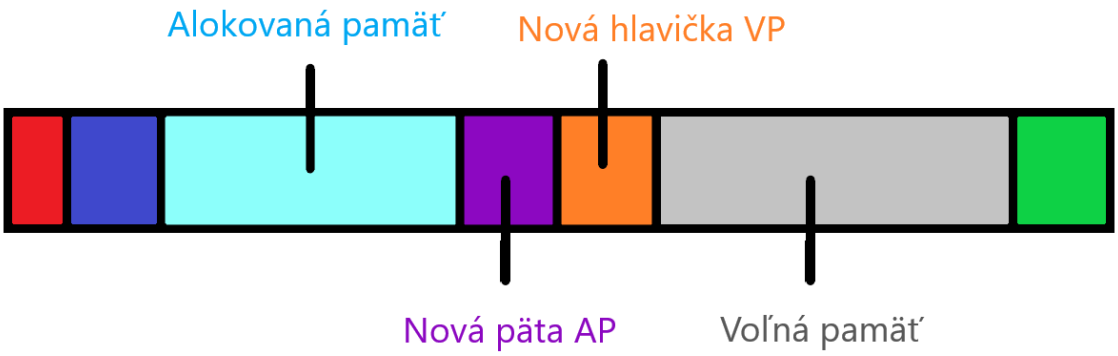
Päta bloku pamäte je identická ako hlavička bloku pamäte (12B).

```
void  memory_init(void* ptr, int size) {

    int globalH = 2 * sizeof(int);
    int header = 3 * sizeof(int);
    int footer = 3 * sizeof(int);

    //kontrola schopnosti inicializacie
    if (size < globalH + header + footer + 8) {
        printf("Initialization ERROR!\n\n");
    }

    else {
        //vynulovanie pamate
        memoryPtr = (char*)(ptr);
        for (int i = 0; i < size; i++) {
            *(memoryPtr + i) = 0;
        }

        //globalna hlavicka pamate
        int* sizeMem = (int*)(memoryPtr);
        *(sizeMem) = size;

        int* freeBlock = (int*)(memoryPtr + sizeof(int));
        *(freeBlock) = 1;
```

Lukáš Michalík, ID:103052
Pondelok 09:00

Funkcia memory_alloc

Funkcia **memory_alloc** má poskytovať služby analogické štandardnému malloc. Teda, jej vstupné parametre sú veľkosť požadovaného súvislého bloku pamäte a funkcia vráti ukazovateľ na úspešne alokovaný kus voľnej pamäte, ktorý sa vyhradil, alebo NULL, keď nie je možné súvislú pamäť požadovanej veľkosť vyhradiť. Pri implementácií som využil metódu “first fit”, a teda, vždy alokujem najbližší vyhovujúci blok pamäte. Vždy prehľadávam všetky dostupné bloky pamäte (voľné/alokované) a keď nájdem blok, ktorý je voľný a jeho veľkosť vyhovuje vstupnému parametru v podobe požadovanej veľkosti. Tak ho alokujem. Pokiaľ sa požadovaná veľkosť zhoduje s veľkosťou bloku, tak ho alokujem celý. Pokiaľ nie, je potrebné tento blok rozdeliť na dva bloky. Jeden alokovaný s požadovanou veľkosťou a druhý voľný, so zvyšnou veľkosťou.




```
void* memory_alloc(int size) {

    char* checkP = (char*)(memoryPtr + (2 * sizeof(int)));
    int* write = (int*)(checkP);
    void* result = NULL;

    bool successfulAllocation = false;
    bool lastBlock = false;
    int reqSize = size + 4 + (6 * sizeof(int));

    //prechadzanie jednotlivych blokov pamate
    for (int i = 0; i < *(int*)(memoryPtr + sizeof(int)); i++) {

        int blockSize = *(int*)(checkP + 0);
        int header = (3 * sizeof(int));
        int footer = (3 * sizeof(int));
```

```
//pozadovana velkost == velkost bloku
if (blockSize < 0 && -(blockSize) == size) {
```

```
//kontrola schopnosti alokacie bloku
else if (blockSize < 0 && (-blockSize) > reqSize) {
```

Lukáš Michalík, ID:103052
Pondelok 09:00

Funkcia memory_check

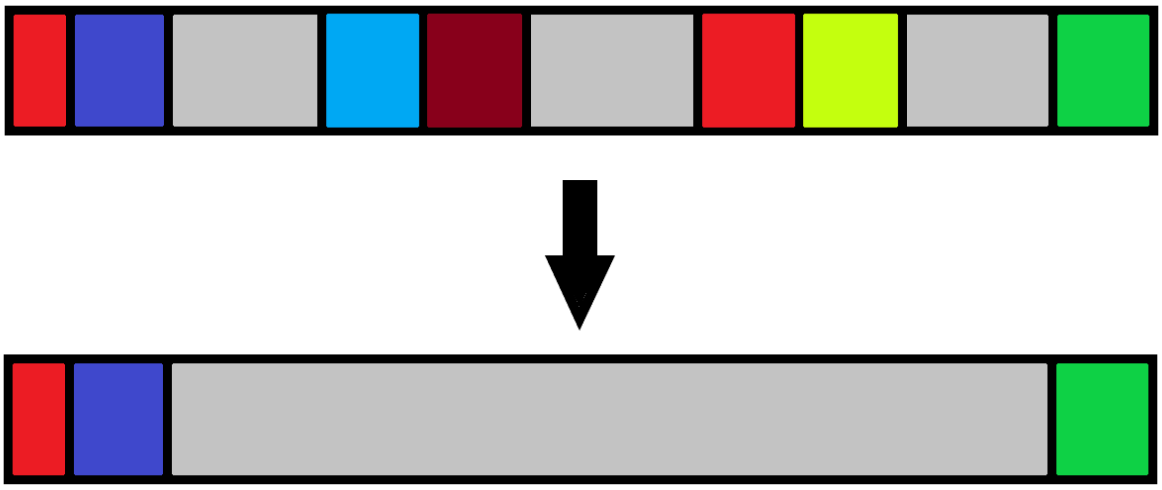
Funkcia **memory_check** slúži na skontrolovanie, či parameter (ukazovateľ) je platný ukazovateľ, ktorý bol v nejakom z predchádzajúcich volaní vrátený funkciou **memory_alloc** a zatiaľ nebol uvoľnený funkciou **memory_free**. Funkcia vráti **0**, ak je ukazovateľ neplatný, inak vráti **1**. Teda v mojom prípade funkcia porovná, či sa daný ukazovateľ nachádza niekde v rozmedzí začiatočnej adresy bloku pamäte a koncovej adresy bloku pamäte. Ak áno, tak prebehne všetky bloky v pamäti a kontroluj, či sa daný ukazovateľ zhoduje s niektorým z blokov. Ak áno, funkcia vráti **1** a ak prebehne všetky bloky a nenájde žiadnu zhodu, ukazovateľ označí za neplatný a funkcia vráti **0.**

```
int memory_check(void* ptr) {

    //posun na zaciatok pamate
    char* checkP = (char*)(memoryPtr + (2 * sizeof(int)));

    //posun na zaciatok hlavicky
    ptr = (void*)((char*)ptr - (3 * sizeof(int)));

    int* memStart = (int*)(memoryPtr + (2 * sizeof(int)));
    int* memEnd = (int*)(memoryPtr + *(int*)(memoryPtr));

    //kontrola rozsahu pamate
    if ((int*)(ptr) < (int*)(memStart) || (int*)(memEnd) < (int*)(ptr)) {
        printf("Pointer is invalid!\n");
        return 0;
    }

    //prehladavanie blokov pamate
    for (int i = 0; i < *(memoryPtr + sizeof(int)); i++) {

        //pointer je platny
        if (ptr == checkP && *(checkP) > 0) {
            printf("Pointer is valid\n");
            return 1;
        }

        //pointer nie je platny, posun na dalsi blok
        else {
            checkP += *(checkP + sizeof(int));
        }
    }
```

Lukáš Michalík, ID:103052
Pondelok 09:00

Funkcia 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. V implementácií som podľa zadania predpokladal, že parameter bude vždy platný ukazovateľ, vrátený z predchádzajúcich volaní funkcie **memory_alloc,** ktorý ešte nebol uvoľnený. Funkcia najskôr vynuluje alokovanú pamäť v bloku, na ktorý ukazuje platný ukazovateľ. Následne skontroluje, či sa pred blokom pamäte nenachádza tiež voľný blok pamäte. Ak áno, tak pomocná funkcia **mergeBlocks** tieto dva bloky spojí do jedného. Následne sa skontroluje aj nasledujúci blok pamäte. V prípade, že je voľný, taktiež sa vykoná spojenie do jedného veľkého bloku. Pokiaľ funkcia úspešne uvoľnila blok pamäte, tak vráti **0**, inak vráti **1**.



```
int memory_free(void* valid_ptr) {

    char* char_valid_ptr = (char*)valid_ptr;
    int* write;
    int header = (3 * sizeof(int));
    int footer = (3 * sizeof(int));

    int nextH = sizeof(int);
    int prevH = sizeof(int);

    int blockSize = *(int*)(char_valid_ptr - header);
    bool successfulFree = false;
    bool merge = false;

    //vynulovanie bloku
    for (int i = 0; i < blockSize; i++) {
        *(char*)(char_valid_ptr + i) = 0;
    }
```

```
//kontrola predchadzajuceho bloku a pripadne spojenie blokov
if ( checkPrevBlock(char_valid_ptr) == 1 ) {
```

```
//kontrola nasledujuceho bloku a pripadne spojenie blokov
if ( checkNextBlock(char_valid_ptr, blockSize) == 1 ) {
```

Lukáš Michalík, ID:103052
Pondelok 09:00

Účinnosť

Nakoľko moje riešenie obsahuje pamäťovo náročnejšie požiadavky na hlavičky a päty, tak pri malom bloku pamäte a jeho rozdeľovaní na malé časti, je toto riešenie nevýhodné. Riešením by mohli byť dynamické hlavičky a päty. Avšak pri vyšších veľkostiach blokov pamäte je tento problém skoro zanedbateľný.