Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №1

з дисципліни «Операційні системи»

Виконала:

студентка гр. ІС-71

Новаківська Катерина

Перевірив:

Дифучин А. Ю.

Київ – 2020

Опис розробленого алгоритму:

Перед виділенням пам’яті заданого розміру робиться перевірка на допустимість вказаного розміру. Шукаємо перший пул, що задовольняє умові, що заданий розмір буде не більшим за розмір пулу. Якщо умова виконується запам’ятовуємо адресу та номер пулу. Пошук припиняється, коли знайдений перший пул. Якщо ж пул не знайдений, то операція виділення пам’яті припиняється, інакше створюється новий блок пам’яті, адреса початку пулу зміщюється в кінець блоку.

Оцінка часу пошуку порожнього блоку пам’яті в найгіршому випадку складе О(Р), де Р – кількість пулів

Оцінка часу вивільнення зайнятого блоку пам’яті складе О(R), де R – кількість правих блоків

Недоліки:

Даний алокатор був розроблений для управління пам’яттю змінної типу char для прикладу, для того, щоб алокатор був універсальним треба його адаптувати до узагальненного типу

Лістинг:

#include <string.h>

#include "stdafx.h"

#include <iostream>

using namespace std;

const int MAX\_POOLS = 1000;

const long BUF\_SIZE = 104888320;

static char ALLBUF[BUF\_SIZE];

static unsigned long AVAIBLE = BUF\_SIZE;

static char\* pools[MAX\_POOLS];

static unsigned int POOLS\_COUNT = 1;

static unsigned int pools\_size[MAX\_POOLS];

static char\* blocks[MAX\_POOLS];

static unsigned int BLOCKS\_COUNT = 0;

static unsigned int block\_size[MAX\_POOLS];

static unsigned int RIGHT\_BLOCK;

static int ARS\_ALLOC\_ERR = 0;

#define NO\_MEMORY 1

#define BLOCK\_NOT\_FOUND 2

void mem\_init(void);

char\* mem\_alloc(unsigned long Size);

void mem\_free(char\*);

int mem\_frag(void);

char\* mem\_realloc(char\*, unsigned long);

void mem\_init(void)

{

pools[0] = ALLBUF;

pools\_size[0] = AVAIBLE;

}

char\* mem\_alloc(unsigned long Size)

{

unsigned long int i, k;

char \*p;

if (Size > AVAIBLE)

{

ARS\_ALLOC\_ERR = NO\_MEMORY;

return 0;

}

p = 0;

for (i = 0; i<POOLS\_COUNT; ++i)

if (Size <= pools\_size[i])

{

p = pools[i];

k = i;

break;

}

if (!p)

{

ARS\_ALLOC\_ERR = NO\_MEMORY;

return 0;

}

blocks[BLOCKS\_COUNT] = p;

block\_size[BLOCKS\_COUNT] = Size;

++BLOCKS\_COUNT;

++RIGHT\_BLOCK;

pools[k] = (char\*)(p + Size + 1);

pools\_size[k] = pools\_size[k] - Size;

AVAIBLE -= Size;

return p;

}

char\* mem\_realloc(char\*p ,unsigned long Size)

{

unsigned long int i, k;

char \*newp;

if (Size > AVAIBLE)

{

ARS\_ALLOC\_ERR = NO\_MEMORY;

return 0;

}

for (i = 0; i<POOLS\_COUNT; ++i)

if (Size <= pools\_size[i])

{

p = pools[i];

k = i;

break;

}

newp = p;

if (!newp)

{

ARS\_ALLOC\_ERR = NO\_MEMORY;

return 0;

}

blocks[BLOCKS\_COUNT] = newp;

block\_size[BLOCKS\_COUNT] = Size;

++BLOCKS\_COUNT;

++RIGHT\_BLOCK;

pools[k] = (char\*)(newp + Size + 1);

pools\_size[k] = pools\_size[k] - Size;

AVAIBLE -= Size;

return newp;

}

void mem\_free(char\* block)

{

unsigned int i, k = 0;

char\* p = 0;

for (i = 0; i<RIGHT\_BLOCK; ++i)

if (block == blocks[i])

{

p = blocks[i];

k = i;

break;

}

if (!p)

{

ARS\_ALLOC\_ERR = BLOCK\_NOT\_FOUND;

}

blocks[k] = 0;

--BLOCKS\_COUNT;

pools[POOLS\_COUNT] = block;

pools\_size[POOLS\_COUNT] = block\_size[k];

++POOLS\_COUNT;

AVAIBLE += block\_size[k];

}

int mem\_frag(void)

{

unsigned int i, k;

char\* p = ALLBUF;

char\* t, \*tmp;

for (i = 0; i<RIGHT\_BLOCK; ++i)

{

t = blocks[i];

if (t == ALLBUF)

{

p = (char\*)(blocks[i] + block\_size[i] + 1);

continue;

}

tmp = p;

for (k = 0, t = blocks[i]; k<block\_size[i]; ++k)

\*p++ = \*t++;

blocks[i] = tmp;

}

POOLS\_COUNT = 1;

pools[0] = p;

AVAIBLE = BUF\_SIZE - (unsigned long)(p - ALLBUF);

pools\_size[0] = AVAIBLE;

RIGHT\_BLOCK = 0;

return 0;

}

int main()

{

char x = 'g';

mem\_init();

char\* p = mem\_alloc(22000);

cout << &p << endl;

char\* newp = mem\_realloc(p, 35000);

cout << &newp << endl;

mem\_frag();

mem\_free(p);

mem\_free(newp);

system("pause");

return 0;

}

Результат виконання:

