Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

**ЗВІТ**

про виконання комп’ютерного практикуму № 1

з дисципліни

“Операційні системи”

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| Проф. Сімоненко В. П. |  | студент 3-го курсу  гр. ІП-51 ФІОТ  Булатов Дмитро Эгоррович |

Київ – 2017

**ЗМІСТ:**

[1 ОПИС АЛГОРИТМУ 3](#_Toc493839366)

[2 Оцінка складності роботи 5](#_Toc493839367)

[3 Оцінка розходів на службову інформацію 6](#_Toc493839368)

[4 Переваги та не доліки розробленого алгоритму 7](#_Toc493839369)

[5 Лістинг програми 8](#_Toc493839370)

[6 Приклад виконання програми 15](#_Toc493839371)

# ОПИС АЛГОРИТМУ

## **Алокація пам’яті**

1. Початок
2. Почати продивлятись пам’ять з початку
3. Якщо поточна область зайнята то переміщаємось до наступної
4. Якщо поточна та наступна область пам’яті вільні то об’єднуємо їх
5. Повторювати крок 3 поки не об’єднаємо останню область пам’яті або поки наступна область не буде зайнятою
6. Якщо поточна область достатньо велика то відділяємо від неї частину пам’яті та повертаємо вказівник на неї
7. Кінець

## **Деалокація памяті**

1. Початок
2. Перевірити чи заданий покажчик не NULL.
3. Перевірити чи заданий покажчик вказує на пам'ять з загального пула та чи зареєстрована дана пам'ять, як зайнята.
4. Помітити область як вільну
5. КІНЕЦЬ

## **Реалокація памяті**

1. ПОЧАТОК
2. Перевірити чи заданий покажчик не NULL.
3. Перевірити чи заданий покажчик вказує на пам'ять з загального пула та чи зареєстрована дана пам'ять, як зайнята.
4. Алокувати нову область пам’яті.
5. Скопіювати в нову область необхідну пам’ять.
6. Деалокувати стару область пам’яті.
7. КІНЕЦЬ

# Оцінка складності роботи

## **Алокація**

Оскільки для зберігання розмірів областей памяті та знаходження вільних областей памяті використовуються однозв’язний список, то виділення пам’яті виконується за O(n)

1. **Деалокація**

Деалокація – це зворотній процес до алокації, і виконується він в 1 операцію, тому швидкість O(1)

1. **Реалокація**

Оскільки реалокація – це алокація нової та де алокація старої памяті то загальна вартість операції O(n)

# Оцінка розходів на службову інформацію

Службова інформація – дескриптори областей

1. Дескриптор – 1 unsigned int

Отже, загальна службова інформація займає 4 байт на 1 область.

# Переваги та не доліки розробленого алгоритму

Розроблений алгоритм дозволяє алокувати та швидко деалокувати необхідні області пам’яті при цьому дескриптор мінімального розміру. До недоліків можна віднести низьку швидкість алокації.

# Лістинг програми

“main.cpp”

#include <iostream>

using namespace std;

typedef unsigned int uint;

const uint highBit = 2147483648;

class Allocator{

void \*memory,\*memoryEnd;

void merge(void\* p){

if (!((\*((uint\*)p)) & highBit)) return;

size\_t memorySize = (\*((uint\*)p)) ^ highBit;

void\* next = p + sizeof(uint) + memorySize;

while (next<memoryEnd &&((\*((uint\*)next)) & highBit)){

size\_t addMemory = ((\*((uint\*)next)) ^ highBit);

memorySize += addMemory + sizeof(uint);

(\*((uint\*)p)) = memorySize | highBit;

next = p + sizeof(uint) + memorySize;

}

}

public:

Allocator(){

memory = new char[4096];

memoryEnd = (char\*)memory+4096;

\*((uint\*)memory) = (4096-sizeof(uint)) | highBit;

}

Allocator(size\_t memorySize){

if (memorySize<5) return;

memory = new char[memorySize];

memoryEnd = (char\*)memory+memorySize;

\*((uint\*)memory) = (memorySize-sizeof(uint)) | highBit;

}

~Allocator(){

delete[] (char\*)memory;

}

void\* allocate(size\_t memorySize){

void\* i=memory;

while (i<memoryEnd)

{

if ((\*((uint\*)i)) & highBit)

{

merge(i);

size\_t blockSize = (\*((uint\*)i)) ^ highBit;

if (blockSize>=memorySize){

if (blockSize > memorySize + sizeof(uint)){

size\_t s2= blockSize - memorySize - sizeof(uint);

\*((uint\*)i) = highBit | memorySize;

void\* j = i + sizeof(uint) + memorySize;

\*((uint\*)j) = highBit | s2;

}

\*((uint\*)i) ^= highBit;

return i+sizeof(uint);

}

}

size\_t blockSize = (\*((uint\*)i)) & (~highBit);

i = i + sizeof(uint) + blockSize;

}

return 0;

}

void free(void\* p){

if (p<memory || p>=memoryEnd) return;

if (((\*((uint\*)p)) & highBit)) return;

p-=sizeof(uint);

size\_t memorySize = (\*((uint\*)p));

(\*((uint\*)p)) ^= highBit;

}

void\* realloc(void\* p, size\_t newSize){

if (p<memory || p>=memoryEnd) return 0;

if (((\*((uint\*)p)) & highBit)) return 0;

p -= sizeof(uint);

int curSize = (\*((uint\*)p));

void\* newPointer = allocate(newSize);

p += sizeof(uint);

for (char \*i=(char\*)p, \*j=(char\*)newPointer;i<p+curSize && i<p+newSize;i++,j++){

\*j = \*i;

}

free(p);

return newPointer;

}

};

int main()

{

Allocator a;

int i=0;

void \*p1=a.allocate(124),\*p2=a.allocate(124);

cout << "124 bytes allocated:" <<p1 << " 124 more bytes allocated:" << p2 << endl;

a.free(p1);

a.free(p2);

void\* p = a.allocate(4092);

cout << "free all memory" << endl;

cout << "max memory allocated:" << p << endl;

a.free(p);

int\* t = (int\*)a.allocate(sizeof(int));

\*t = 12345;

cout << "data before realloc:" << \*t << " pointer:" << t << endl;

t = (int\*)a.realloc(t,sizeof(int));

cout << "data after realloc:" << \*t << " pointer:" << t << endl;

return 0;

}

# Приклад виконання програми

