**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизації систем обробки інформації та управління**

**“Сучасні операційні системи”**

**Лабораторна робота №2**

**Перевірив: Дифучин А. Ю.**

**Виконав: Загинайло Є.О.**

**Студент гр. ІС-73, ФІОТ,**

**3 курс**

Київ

НТУУ «КПІ ім. І. Сікорського»

2020

**Отчёт**

**Описание алгоритма:**

Все блоки памяти делятся на две группы:

* Блоки с размерами меньше или равными половине страницы.
* Остальные (1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256)

Размер страницы, 4Кбайт, заголовок – 20 байт

mem\_aloc(size\_t size) виділення пам’яті:

Якщо розмір більший ніж потрібно, блок розбивається на зайнятий і вільний, і користувачу повертається \*ptr на початок зайнятого блоку. Якщо потрібний блок не знайдений, то повертається NULL.

mem\_free(void\* ptr) звільнення виділеного блоку пам’яті:

Блок помічається вільним та об’єднується з сусідніми вільними блоками.

mem\_realloc(void\* ptr, size\_t size) перевиділення пам’яті:

Якщо addr = NULL, то виконується виклик mem\_alloc(size).

Якщо пам'яті не вистачить, то старий блок пам'яті не видалиться і нульовий \*ptr буде повернений.

В функції виділяється нова область пам'яті розміром в size байт, копіювання туди перших size байт старої області та звільнення старого блоку пам'яті.

**Оцінка часу пошуку вільного блоку пам’яті**:

O(log(n)), де n – кількість блоків у пам’яті.

**Оцінка часу звільнення занятого блоку**:

O(1).

**Оцінка витрати пам'яті для зберігання службовох інформації:**

20 \* (кількість блоків)

**Переваги:**

1. Малий час виконання звільнення блоку.
2. Швидке виділення блоку

**Недоліки:**

1. Складніша реалізація

**Лістинг на мові С:**

**Allocator.cpp**

**#include "StdAfx.h"**

**#include "Allocator.h"**

**#include <iostream>**

**#include <stdlib.h>**

**#include <ctime>**

**using namespace std;**

**struct memoryHeader**

**{**

**void\* page16;**

**size\_t page16Count;**

**void\* page32;**

**size\_t page32Count;**

**void\* page64;**

**size\_t page64Count;**

**void\* page128;**

**size\_t page128Count;**

**void\* page256;**

**size\_t page256Count;**

**void\* page512;**

**size\_t page512Count;**

**void\* page1024;**

**size\_t page1024Count;**

**void\* page2048;**

**size\_t page2048Count;**

**void\* pageFull;**

**size\_t pageFullCount;**

**};**

**struct pageHeader**

**{**

**char status;**

**size\_t blockSize;**

**size\_t freeBlocksCount;**

**void\* freeBlock;**

**void\* nextPage;**

**};**

**Allocator::Allocator(size\_t pageCount)**

**{**

**this->pageCount = pageCount;**

**if (this->pageCount < 9)**

**{**

**this->pageCount = 9;**

**}**

**size\_t memorySize = (pageSize + sizeof(pageHeader)) \* this->pageCount + sizeof(memoryHeader);**

**memory = new char[memorySize];**

**struct memoryHeader\* mHeader = (memoryHeader\*) memory;**

**mHeader->page16 = NULL;**

**mHeader->page16Count = NULL;**

**mHeader->page32 = NULL;**

**mHeader->page32Count = NULL;**

**mHeader->page64 = NULL;**

**mHeader->page64Count = NULL;**

**mHeader->page128 = NULL;**

**mHeader->page128Count = NULL;**

**mHeader->page256 = NULL;**

**mHeader->page256Count = NULL;**

**mHeader->page512 = NULL;**

**mHeader->page512Count = NULL;**

**mHeader->page1024 = NULL;**

**mHeader->page1024Count = NULL;**

**mHeader->page2048 = NULL;**

**mHeader->page2048Count = NULL;**

**mHeader->pageFull = (void\*) ((size\_t) memory + sizeof(memoryHeader));**

**mHeader->pageFullCount = this->pageCount;**

**struct pageHeader\* pHeader = (pageHeader\*) (mHeader->pageFull);**

**for (size\_t i = 0; i < this->pageCount; i++)**

**{**

**pHeader->status = 0;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->freeBlocksCount = 1;**

**pHeader->blockSize = pageSize;**

**if (i < (this->pageCount - 1))**

**{**

**pHeader->nextPage = (void\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize);**

**}**

**else**

**{**

**pHeader->nextPage = NULL;**

**}**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**}**

**Allocator::~Allocator(void)**

**{**

**delete[] memory;**

**}**

**void\* Allocator::mem\_alloc(size\_t size)**

**{**

**struct memoryHeader\* mHeader = (memoryHeader\*) memory;**

**struct pageHeader\* pHeader = NULL;**

**if (size <= (pageSize / 2))**

**{**

**if (size > (pageSize / 4))**

**{**

**if (mHeader->page2048 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page2048;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(4096);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page2048;**

**pHeader->blockSize = 2048;**

**pHeader->freeBlocksCount = pageSize / 2048;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page2048 = (void\*) pHeader;**

**mHeader->page2048Count = mHeader->page2048Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**if (mHeader->pageFullCount == 0)**

**{**

**return NULL;**

**}**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page2048;**

**pHeader->blockSize = 2048;**

**pHeader->freeBlocksCount = pageSize / 2048;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page2048 = (void\*) pHeader;**

**mHeader->page2048Count = mHeader->page2048Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (size > (pageSize / 8))**

**{**

**if (mHeader->page1024 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page1024;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(2048);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page1024;**

**pHeader->blockSize = 1024;**

**pHeader->freeBlocksCount = pageSize / 1024;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page1024 = (void\*) pHeader;**

**mHeader->page1024Count = mHeader->page1024Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page1024;**

**pHeader->blockSize = 1024;**

**pHeader->freeBlocksCount = pageSize / 1024;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page1024 = (void\*) pHeader;**

**mHeader->page1024Count = mHeader->page1024Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (size > (pageSize / 16))**

**{**

**if (mHeader->page512 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page512;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(1024);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page512;**

**pHeader->blockSize = 512;**

**pHeader->freeBlocksCount = pageSize / 512;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page512 = (void\*) pHeader;**

**mHeader->page512Count = mHeader->page512Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page512;**

**pHeader->blockSize = 512;**

**pHeader->freeBlocksCount = pageSize / 512;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page512 = (void\*) pHeader;**

**mHeader->page512Count = mHeader->page512Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (size > (pageSize / 32))**

**{**

**if (mHeader->page256 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page256;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(512);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page256;**

**pHeader->blockSize = 256;**

**pHeader->freeBlocksCount = pageSize / 256;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page256 = (void\*) pHeader;**

**mHeader->page256Count = mHeader->page256Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page256;**

**pHeader->blockSize = 256;**

**pHeader->freeBlocksCount = pageSize / 256;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page256 = (void\*) pHeader;**

**mHeader->page256Count = mHeader->page256Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (size > (pageSize / 64))**

**{**

**if (mHeader->page128 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page128;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(256);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page128;**

**pHeader->blockSize = 128;**

**pHeader->freeBlocksCount = pageSize / 128;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page128 = (void\*) pHeader;**

**mHeader->page128Count = mHeader->page128Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page128;**

**pHeader->blockSize = 128;**

**pHeader->freeBlocksCount = pageSize / 128;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page128 = (void\*) pHeader;**

**mHeader->page128Count = mHeader->page128Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (size > (pageSize / 128))**

**{**

**if (mHeader->page64 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page64;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(128);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page64;**

**pHeader->blockSize = 64;**

**pHeader->freeBlocksCount = pageSize / 64;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page64 = (void\*) pHeader;**

**mHeader->page64Count = mHeader->page64Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page64;**

**pHeader->blockSize = 64;**

**pHeader->freeBlocksCount = pageSize / 64;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page64 = (void\*) pHeader;**

**mHeader->page64Count = mHeader->page64Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (size > (pageSize / 256))**

**{**

**if (mHeader->page32 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page32;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(64);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page32;**

**pHeader->blockSize = 32;**

**pHeader->freeBlocksCount = pageSize / 32;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page32 = (void\*) pHeader;**

**mHeader->page32Count = mHeader->page32Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page32;**

**pHeader->blockSize = 32;**

**pHeader->freeBlocksCount = pageSize / 32;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page32 = (void\*) pHeader;**

**mHeader->page32Count = mHeader->page32Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**else**

**{**

**if (mHeader->page16 != NULL)**

**{**

**pHeader = (pageHeader\*) mHeader->page16;**

**while ((pHeader->freeBlock == NULL) && (pHeader->nextPage != NULL))**

**{**

**pHeader = (pageHeader\*) pHeader->nextPage;**

**}**

**if (pHeader->freeBlock != NULL)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) pHeader->freeBlock;**

**pHeader->freeBlock = (void\*) \*nextBlockPtr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount - 1;**

**return (void\*) nextBlockPtr;**

**}**

**else**

**{**

**if (mHeader->pageFull == NULL)**

**{**

**return mem\_alloc(32);**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page16;**

**pHeader->blockSize = 16;**

**pHeader->freeBlocksCount = pageSize / 16;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page16 = (void\*) pHeader;**

**mHeader->page16Count = mHeader->page16Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**pHeader->status = 1;**

**mHeader->pageFull = pHeader->nextPage;**

**pHeader->nextPage = mHeader->page16;**

**pHeader->blockSize = 16;**

**pHeader->freeBlocksCount = pageSize / 16;**

**size\_t\* blockPtr = (size\_t\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize - pHeader->blockSize);**

**size\_t nextBlock = NULL;**

**while ((size\_t) blockPtr >= ((size\_t) pHeader + sizeof(pageHeader)))**

**{**

**\*blockPtr = nextBlock;**

**nextBlock = (size\_t) blockPtr;**

**blockPtr = (size\_t\*) ((size\_t) blockPtr - pHeader->blockSize);**

**}**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**mHeader->pageFullCount = mHeader->pageFullCount - 1;**

**mHeader->page16 = (void\*) pHeader;**

**mHeader->page16Count = mHeader->page16Count + 1;**

**return mem\_alloc(size);**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**else**

**{**

**size\_t neededPages = size / pageSize;**

**if ((size % pageSize) != 0)**

**{**

**neededPages++;**

**}**

**if (mHeader->pageFullCount < neededPages)**

**{**

**return NULL;**

**}**

**else**

**{**

**pHeader = (pageHeader\*) mHeader->pageFull;**

**struct pageHeader\* nextPageHeader = pHeader;**

**for (size\_t i = 0; i < neededPages; i++)**

**{**

**nextPageHeader->status = 2;**

**nextPageHeader->freeBlock = NULL;**

**nextPageHeader->freeBlocksCount = neededPages;**

**nextPageHeader = (pageHeader\*) nextPageHeader->nextPage;**

**}**

**mHeader->pageFull = (void\*) nextPageHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount - neededPages;**

**return (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**}**

**}**

**}**

**void Allocator::mem\_free(void\* ptr)**

**{**

**if (ptr == NULL)**

**{**

**return;**

**}**

**struct memoryHeader\* mHeader = (memoryHeader\*) memory;**

**struct pageHeader\* pHeader = (pageHeader\*) ((size\_t) memory + sizeof(memoryHeader));**

**while (((size\_t) ptr < (size\_t) pHeader) || ((size\_t) ptr > ((size\_t) pHeader + sizeof(pageHeader) + pageSize)))**

**{**

**pHeader = (pageHeader\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize);**

**}**

**if (pHeader->status == 2)**

**{**

**struct pageHeader\* pagePtr = pHeader;**

**size\_t pagePtrCount = pHeader->freeBlocksCount;**

**for (size\_t i = 0; i < pagePtrCount; i++)**

**{**

**pagePtr->status = 0;**

**pagePtr->freeBlocksCount = 1;**

**pagePtr->freeBlock = (void\*) ((size\_t) pagePtr + sizeof(pageHeader));**

**if (i < (pagePtrCount - 1))**

**{**

**pagePtr = (pageHeader\*) pagePtr->nextPage;**

**}**

**}**

**pagePtr->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + pagePtrCount;**

**}**

**else**

**{**

**if (pHeader->status == 1)**

**{**

**size\_t\* nextBlockPtr = (size\_t\*) ptr;**

**\*nextBlockPtr = (size\_t) pHeader->freeBlock;**

**pHeader->freeBlock = ptr;**

**pHeader->freeBlocksCount = pHeader->freeBlocksCount + 1;**

**if (pHeader->freeBlocksCount == (pageSize / pHeader->blockSize))**

**{**

**if (pHeader->blockSize == 2048)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page2048;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page2048 = pHeader->nextPage;**

**mHeader->page2048Count = mHeader->page2048Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page2048Count = mHeader->page2048Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**if (pHeader->blockSize == 1024)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page1024;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page1024 = pHeader->nextPage;**

**mHeader->page1024Count = mHeader->page1024Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page1024Count = mHeader->page1024Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**if (pHeader->blockSize == 512)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page512;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page512 = pHeader->nextPage;**

**mHeader->page512Count = mHeader->page512Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page512Count = mHeader->page512Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**if (pHeader->blockSize == 256)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page256;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page256 = pHeader->nextPage;**

**mHeader->page256Count = mHeader->page256Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page256Count = mHeader->page256Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**if (pHeader->blockSize == 128)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page128;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page128 = pHeader->nextPage;**

**mHeader->page128Count = mHeader->page128Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page128Count = mHeader->page128Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**if (pHeader->blockSize == 64)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page64;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page64 = pHeader->nextPage;**

**mHeader->page64Count = mHeader->page64Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page64Count = mHeader->page64Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**if (pHeader->blockSize == 32)**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page32;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page32 = pHeader->nextPage;**

**mHeader->page32Count = mHeader->page32Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page32Count = mHeader->page32Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**else**

**{**

**struct pageHeader\* pagePtr = (pageHeader\*) mHeader->page16;**

**if ((size\_t) pHeader == (size\_t) pagePtr)**

**{**

**mHeader->page16 = pHeader->nextPage;**

**mHeader->page16Count = mHeader->page16Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**else**

**{**

**while ((size\_t) pHeader != (size\_t) (pagePtr->nextPage))**

**{**

**pagePtr = (pageHeader\*) pHeader->nextPage;**

**}**

**pagePtr->nextPage = pHeader->nextPage;**

**mHeader->page16Count = mHeader->page16Count - 1;**

**pHeader->freeBlock = (void\*) ((size\_t) pHeader + sizeof(pageHeader));**

**pHeader->status = 0;**

**pHeader->blockSize = pageSize;**

**pHeader->freeBlocksCount = 1;**

**pHeader->nextPage = mHeader->pageFull;**

**mHeader->pageFull = (void\*) pHeader;**

**mHeader->pageFullCount = mHeader->pageFullCount + 1;**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**void\* Allocator::mem\_realloc(void\* ptr, size\_t size)**

**{**

**if (ptr == NULL)**

**{**

**return mem\_alloc(size);**

**}**

**struct memoryHeader\* mHeader = (memoryHeader\*) memory;**

**struct pageHeader\* pHeader = (pageHeader\*) ((size\_t) memory + sizeof(memoryHeader));**

**while (((size\_t) ptr < (size\_t) pHeader) || ((size\_t) ptr > ((size\_t) pHeader + sizeof(pageHeader) + pageSize)))**

**{**

**pHeader = (pageHeader\*) ((size\_t) pHeader + sizeof(pageHeader) + pageSize);**

**}**

**if (((pHeader->blockSize / 2) < size) && (pHeader->blockSize >= size))**

**{**

**return ptr;**

**}**

**void\* resultPtr = mem\_alloc(size);**

**if (resultPtr == NULL)**

**{**

**return NULL;**

**}**

**char\* ptr1 = (char\*) ptr;**

**char\* ptr2 = (char\*) resultPtr;**

**if ((pHeader->status == 1) || ((pHeader->status == 2) && (pHeader->freeBlocksCount == 1)) || ((pHeader->status == 2) && (size <= pageSize)))**

**{**

**size\_t copySize;**

**if (size < pHeader->blockSize)**

**{**

**copySize = size;**

**}**

**else**

**{**

**copySize = pHeader->blockSize;**

**}**

**for (size\_t i = 0; i < copySize; i++)**

**{**

**\*ptr2 = \*ptr1;**

**ptr1++;**

**ptr2++;**

**}**

**}**

**else**

**{**

**struct pageHeader\* copyPHeader = (pageHeader\*) ((size\_t) resultPtr - sizeof(pageHeader));**

**size\_t copyPagesCount;**

**if (pHeader->freeBlocksCount < copyPHeader->freeBlocksCount)**

**{**

**copyPagesCount = pHeader->freeBlocksCount;**

**}**

**else**

**{**

**copyPagesCount = copyPHeader->freeBlocksCount;**

**}**

**for (size\_t i = 0; i < copyPagesCount; i++)**

**{**

**for (size\_t j = 0; j < pageSize; j++)**

**{**

**\*ptr2 = \*ptr1;**

**ptr1++;**

**ptr2++;**

**}**

**ptr1 = (char\*) ((size\_t) pHeader->nextPage + sizeof(pageHeader));**

**ptr2 = (char\*) ((size\_t) copyPHeader->nextPage + sizeof(pageHeader));**

**}**

**}**

**mem\_free(ptr);**

**return resultPtr;**

**}**

**size\_t Allocator::getPageSize()**

**{**

**return pageSize;**

**}**

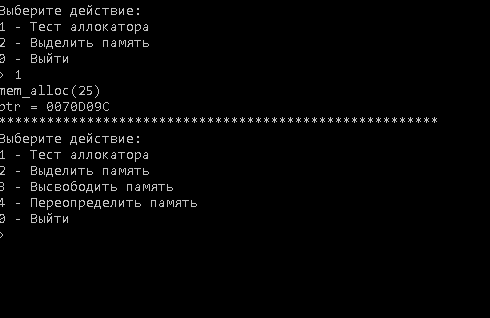
**size\_t Allocator::getPageCount()**

**{**

**return pageCount;**

**}**

Пример работы алокатора:



**Висновок:**

В результаті виконання лабораторної роботи було розроблено програму алокатора пам'яті загального призначення. Реалізовано mem\_alloc(), mem\_realloc(), mem\_free(), що складають інтерфейс алокатора пам'яті загального призначення.