**Лабораторна робота №2**

з курсу «Сучасні операційні системи»

Виконала:

студентка 3 курсу ФІОТ

групи ІС-73

Мигуля Ірина

Перевірив:

Дифучин А.Ю.

**Аллокатор пам’яті загального призначення**

(Частина 2)

***Тема:*** аллокатор пам’яті загального призначення.

***Мета:*** розробити аллокатор загального призначення.

**Лістинг програми**

#include "allocator.h"

mem\_page\* global\_memory;

size\_t page\_count;

void mem\_dump()

{

printf("------------Memory Dump------------\n");

printf("Global Memory Address: %p\n", global\_memory);

for (size\_t i = 0; i < page\_count; i++)

{

printf("-----------------------------\n");

printf("Page: %i State: %3i Address: %p\n", i, global\_memory[i].state, global\_memory[i].address);

printf("Data: %s\n", (char\*)global\_memory[i].address);

if (global\_memory[i].state == BLOCK || global\_memory[i].state == DIVIDE)

printf("Data: %s\n", (char\*)((size\_t)global\_memory[i].address + PAGE\_SIZE / 2));

printf("-----------------------------\n");

}

}

void init\_memory(size\_t size)

{

global\_memory = (mem\_page\*)malloc(size);

global\_memory = (mem\_page\*)calloc(MEMORY\_SIZE, 1);

page\_count = size / (PAGE\_SIZE + PAGE\_INFO);

for (size\_t i = 0; i < page\_count; i++)

{

global\_memory[i].address = (void\*)((unsigned long int)global\_memory + page\_count \* PAGE\_INFO + i \* PAGE\_SIZE);

global\_memory[i].size = 0;

global\_memory[i].state = FREE;

}

}

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

{

void\* address;

size\_t counter = 0;

size\_t firstpage = 0;

size\_t pages\_alloc = (double)size / PAGE\_SIZE + 0.5;

size\_t i;

if (size <= PAGE\_SIZE)

{

for (i = 0; i < page\_count; i++)

{

if (global\_memory[i].state == FREE || (global\_memory[i].state == DIVIDE && size <= PAGE\_SIZE / 2))

{

address = (void\*)((size\_t)(global\_memory[i].address) + (global\_memory[i].state == DIVIDE ? PAGE\_SIZE / 2 : 0));

if (size > PAGE\_SIZE / 2)

{

global\_memory[i].state = BLOCK;

global\_memory[i].size = 1;

}

else

{

global\_memory[i].state = global\_memory[i].state == DIVIDE ? BLOCK : DIVIDE;

global\_memory[i].size = 0;

}

return address;

}

}

}

else

{

for (i = 0; i < page\_count; i++)

{

if (global\_memory[i].state == FREE)

{

counter += 1;

firstpage = i - counter + 1;

if (counter == pages\_alloc)

{

for (size\_t j = firstpage; j < firstpage + counter; j++)

{

global\_memory[j].state = MULTBLOCK;

}

global\_memory[firstpage].size = pages\_alloc \* PAGE\_SIZE;

address = global\_memory[firstpage].address;

return address;

}

}

else

{

counter = 0;

firstpage = 0;

}

}

}

return NULL;

}

void\* mem\_realloc(void\* address, size\_t size)

{

void\* new\_address;

size\_t size\_of\_block = (size\_t)address - (size\_t)global\_memory - PAGE\_INFO \* page\_count;

size\_t started\_block\_page = size\_of\_block / PAGE\_SIZE;

if (global\_memory[started\_block\_page].state == MULTBLOCK)

size\_of\_block = global\_memory[started\_block\_page].size;

else if (global\_memory[started\_block\_page].size != 0)

size\_of\_block = global\_memory[started\_block\_page].size \* PAGE\_SIZE;

else

size\_of\_block = PAGE\_SIZE / 2;

new\_address = mem\_alloc(size);

if (new\_address)

{

memmove(new\_address, address, size\_of\_block > size ? size : size\_of\_block);

mem\_free(address);

return new\_address;

}

return NULL;

}

void mem\_free(void\* address)

{

size\_t size\_of\_block = (size\_t)address - (size\_t)global\_memory - PAGE\_INFO \* page\_count;

size\_t started\_block\_page = size\_of\_block / PAGE\_SIZE;

size\_t page\_state = global\_memory[started\_block\_page].state;

if (page\_state == DIVIDE)

global\_memory[started\_block\_page].state = FREE;

else if (page\_state == BLOCK)

{

if (global\_memory[started\_block\_page].size == 0)

global\_memory[started\_block\_page].state = DIVIDE;

else

global\_memory[started\_block\_page].state = FREE;

}

else if (page\_state == MULTBLOCK)

for (size\_t i = started\_block\_page; i <global\_memory[started\_block\_page].size / PAGE\_SIZE; i++)

global\_memory[i].state = FREE;

std::memset(address, 0, global\_memory[started\_block\_page].size == 0 ? PAGE\_SIZE / 2: global\_memory[started\_block\_page].size);

global\_memory[started\_block\_page].size = 0;

address = NULL;

}

/