Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІКТА

Кафедра СКС



Лабораторна робота № 3

З дисципліни «Системне програмне забезпечення*»*

На тему: «Керування оперативною пам’яттю»

Виконала:

ст. гр. КІ-307

Ващишин І.І.

Прийняв:

ст. викладач

Олексів М. В.

Львів – 2025

**Мета:** Навчитися керувати оперативною пам’яттю в середовищі операційної системи, розробляти програми керування пам’яттю.

**Теоретичні відомості**

**Поняття віртуальної пам’яті**

Віртуальна пам'ять —технологія, в якій вводиться рівень додаткових перетворень між адресами пам'яті, використовуваних процесом, і адресами фізичної пам'яті комп'ютера. Перетворення забезпечують захист пам'яті та відсутність прив'язки процесу до адрес фізичної пам'яті. Завдяки віртуальній пам'яті фізична пам'ять адресного простору процесу може бути фрагментованою, оскільки основний обсяг пам'яті, яку займає процес, більшу частину часу залишається вільним. Адреси можна переміщати так, щоб основній пам'яті відповідали тільки ті розділи адресного простору процесу, які використовуються у конкретний момент.

Невикористовувані розділи адресного простору ставляться у відповідність повільнішій пам'яті (простору на жорсткому диску), а в цей час інші процеси використовують основну пам'ять. Коли ж розділ знадобиться, його дані завантажують з диска в основну пам'ять. Дані зчитуються з диска в основну пам'ять під час звертання до них.

У такий спосіб можна збільшити розмір адресного простору процесу і забезпечити виконання процесів, що за розміром перевищують основну пам'ять.

**Логічна і фізична адресація пам'яті**

Логічна або віртуальна адреса — адреса, яку генерує програма, запущена на деякому процесорі. Адреси, що використовують інструкції конкретного процесора, є логічними адресами. Сукупність логічних адрес становить логічний адресний простір.

Фізична адреса - адреса, якою оперує мікросхема пам'яті. Прикладна програма в сучасних комп'ютерах не має справи з фізичними адресами. Спеціальний апаратний пристрій MMU (memory management unit - пристрій керування пам'яттю) відповідає за перетворення логічних адрес у фізичні. Сукупність усіх доступних фізичних адрес.

**Особливості сегментації пам'яті**

Сегментація пам'яті зображає логічний адресний простір як сукупність незалежних блоків змінної довжини, які називають сегментами. Кожен сегмент містить дані одного призначення.

Кожен сегмент має ім'я і довжину. Логічна адреса складається з номера сегмента і зсуву всередині сегмента; з такими адресами працює прикладна програма. Компілятори створюють окремі сегменти для різних даних програми (сегмент коду, сегмент даних, сегмент стека). Під час завантаження програми у пам'ять створюють таблицю дескрипторів сегментів процесу, кожний елемент якої відповідає одному сегменту і складається із базової адреси, значення межі та прав доступу.

Під час формування адреси її сегментна частина вказує на відповідний елемент таблиці дескрипторів сегментів процесу.

Якщо зсув більший, ніж задане значення межі (або якщо права доступу процесу не відповідають правам, заданим для сегмента), то апаратне забезпечення генерує помилку. Коли ж усе гаразд, сума бази і зсуву в разі чистої сегментації дасть у результаті фізичну адресу в основній пам'яті. Якщо сегмент вивантажений на диск, спроба доступу до нього спричиняє його завантаження з диска в основну пам'ять. Кожному сегменту відповідає неперервний блок пам'яті такої самої довжини, що перебуває в довільному місці фізичної пам'яті або на диску.

**Реалізація сегментації**

Логічні адреси в програмі формуються з використанням сегментації і мають такий вигляд:“селектор-зсув”. Значення селектора завантажуютьу спеціальний регістр процесора (сегментний регістр) і використовують як індекс у таблиці дескрипторів сегментів, що перебуває в пам'яті та є аналогом таблиці сегментів.

Селектор містить індекс дескриптора в таблиці, біт індикатора локальної або глобальної таблиці та необхідний рівень привілеїв.

Для системи задають спільну глобальну таблицю дескрипторів (Global Descriptor Table, GDT), а для кожної задачі - локальну таблицю дескрипторів (Local Descriptor Table, LDT).

Дескриптори мають довжину 64 біти. Вони визначають властивостіпрограмних об'єктів (наприклад, сегментів пам'яті або таблиць дескрипторів).

Дескриптор містить значення бази, яке відповідає адресі об'єкта (початок сегмента); значення межі; тип об'єкта (сегмент, таблиця дескрипторів тощо); характеристики захисту.

Звертання до таблиць дескрипторів підтримується апаратно. Якщо задані в дескрипторі характеристики захисту не відповідають рівню привілеїв, визначеному селектором, отримати доступ до пам'яті за його допомогою буде неможливо. Так забезпечують захист пам'яті.

Для архітектури внаслідок перетворення логічної адреси отримують нефізичну адресу, а вид адреси, який називають лінійною адресою.

Сторінкова організація пам'яті

До основних технологій реалізації віртуальної пам'яті належить сторінкова організація пам'яті. Її головна ідея — розподіл пам'яті блоками фіксованої довжини, що називають сторінками.

Базові принципи сторінкової організації пам'яті

Фізичну пам'ять розбивають на блоки фіксованої довжини — фрейми, або сторінкові блоки. Логічну пам'ять, у свою чергу, розбивають на блоки такої самої довжини — сторінки. Коли процес починає виконуватися, його сторінки завантажуються в доступні фрейми фізичної пам'яті з диска або іншого носія.

Сторінкова організація пам'яті має апаратну підтримку. Кожна адреса, яку генерує процесор, ділиться на дві частини: номер сторінки і зсув сторінки. Номер сторінки використовують як індекс у таблиці сторінок.

**Основні функції для роботи з оперативною пам’яттю:**

**VirtualAlloc()** – функція, яка виділяє пам’ять у віртуальному адресному просторі.

**VirtualFree()** – функція, яка звільняє попередньо виділену пам’ять.

**VirtualLock()** – блокує сторінки пам’яті в оперативній пам’яті.

**VirtualUnlock()** – розблоковує сторінки пам’яті.

**SetProcessWorkingSetSize()** – використовується для встановлення мінімального та максимального розміру робочого набору поля.

**GetProcessWorkingSetSize()** – для отримання мінімального та максимального розміру робочого набору поля.

**Хід виконання роботи**

**Варіант 3:** Розробити програму, яка демонструє управління структурою даних типу «стек», елементами якого є значення типу char. Стек реалізувати за допомогою динамічного масиву розміром 12 Кб., пам’ять під який виділити за допомогою функції VirtualAlloc(). Операції, що виконуються над стеком: •перевірити, стек порожній чи не порожній; •заштовхнути елемент; •виштовхнути елемент; •переглянути вершину стека; •визначити скільки елементів у стеку.

**Код програми:**

#include <iostream>

#include <Windows.h>

#define STACK\_CAPACITY\_BYTE 12288

#define STACK\_CAPACITY (STACK\_CAPACITY\_BYTE / sizeof(char))

typedef struct {

char\* data;

size\_t size;

size\_t capacity;

} STACK;

void CreateStack(STACK\* stack) {

stack->data = (char\*)VirtualAlloc(NULL, STACK\_CAPACITY\_BYTE, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (!stack->data) {

std::cerr << "Failed to allocate memory for stack!\n";

exit(1);

}

stack->size = 0;

stack->capacity = STACK\_CAPACITY;

}

bool isEmpty(STACK\* stack) {

return stack->size == 0;

}

bool isFull(STACK\* stack) {

return stack->size == stack->capacity;

}

void push(STACK\* stack, char value) {

if (isFull(stack)) {

std::cout << "Stack is full!\n";

return;

}

stack->data[stack->size++] = value;

}

char pop(STACK\* stack) {

if (isEmpty(stack)) {

std::cout << "Stack is empty!\n";

return '\0';

}

char popped = stack->data[--stack->size];

std::cout << "Popped element: " << popped << "\n";

return popped;

}

char top(STACK\* stack) {

if (isEmpty(stack)) {

std::cout << "Stack is empty!\n";

return '\0';

}

return stack->data[stack->size - 1];

}

size\_t getSize(STACK\* stack) {

return stack->size;

}

void showStack(STACK\* stack) {

if (isEmpty(stack)) {

std::cout << "Stack is empty!\n";

return;

}

std::cout << "Stack content (top to bottom): ";

for (int i = stack->size - 1; i >= 0; --i) {

std::cout << stack->data[i] << ' ';

}

std::cout << '\n';

}

void deleteStack(STACK\* stack) {

if (stack->data) {

VirtualFree(stack->data, 0, MEM\_RELEASE);

}

}

int main() {

STACK myStack{};

CreateStack(&myStack);

std::cout << (isEmpty(&myStack) ? "Stack is empty.\n" : "Stack is not empty.\n");

push(&myStack, 'A');

push(&myStack, 'B');

push(&myStack, 'C');

showStack(&myStack);

std::cout << "Current top of stack: " << top(&myStack) << "\n";

pop(&myStack);

std::cout << "New top of stack: " << top(&myStack) << "\n";

showStack(&myStack);

deleteStack(&myStack);

return 0;

}

**Результат виконання:**

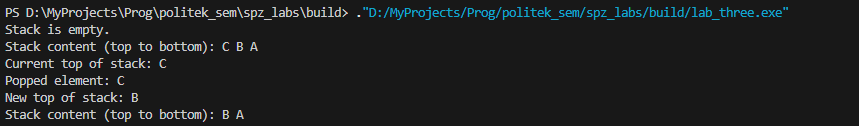


Рис. 1. Результат виконання програми.

**Висновок:** на даній лабораторній роботі я навчилася керувати оперативною пам’яттю в середовищі операційної системи, також я розробила програму яка демонструє, як можна виділяти пам'ять в адресному просторі.