1. МИНОБРНАУКИ РОССИИ
2. САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
3. ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
4. «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
5. Кафедра Вычислительной техники

ОТЧЁТ

по лабораторной работе №2

1. по дисциплине «Операционные системы»
2. Тема: Управление памятью

| Студент гр. 3312 |  | Нкусси ф.М |
| --- | --- | --- |
| Преподаватель |  | Тимофеев А.В. |

**Оглавление**

[1. Введение](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214820) 3

[2. Управление дисками, каталогами и файлами 5](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214821)

[2.1.](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214822) **Вывод списка дисков** 5

[2.2. Вывод информации о диске и размер свободного пространства 5](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214823)

[2.3. Создание и удаление заданных каталогов 5](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214824)

[2.4. Создание файлов в новых каталогах 6](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214825)

[2.5. Копирование и перемещение файлов между каталогами 7](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214826)

[2.6. Анализ и изменение атрибутов файлов 9](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214827)

[2.7. Исходный код программы 24](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214828)

[2.8. Выводы 27](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214829)

[3. Копирование файла с помощью перекрывающих операций ввода-вывода 28](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214830)

[3.1. Создание и запуск консольного приложения 28](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214831)

[3.2. Проверка приложения на разных размерах копируемых блоков 29](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214832)

[3.3. Проверка приложения на разном числе операция ввода-вывода 30](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214833)

[3.4. Исходный код программы 31](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214834)

[3.5. Выводы 36](http://../../../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx#_Toc84214835)

1. **1. Введение**

Тема работы: Управление памятью.

Цель работы: исследовать механизмы управления виртуальной

памятью Win32.

**Указания к выполнению**

**Задание 2.1**. Исследовать виртуальное адресное пространство

процесса.

1. Создайте консольное приложение с меню (каждая выполняемая

функция и/или операция должна быть доступна по отдельному пункту

меню), которое выполняет:

− получение информации о вычислительной системе (функция

Win32 API – GetSystemInfo);

− определение статуса виртуальной памяти (функция Win32 API –

GlobalMemoryStatus);

− определение состояния конкретного участка памяти по заданному

с клавиатуры адресу (функция Win32 API – VirtualQuery); −

раздельное резервирование региона и передачу ему физической

памяти в автоматическом режиме и в режиме ввода адреса начала

региона (функция Win32 API – VirtualAlloc, VirtualFree); −

одновременное резервирование региона и передача ему физической

памяти в автоматическом режиме и в режиме ввода адреса начала

региона (функция Win32 API – VirtualAlloc, VirtualFree);

− запись данных в ячейки памяти по заданным с клавиатуры

адресам;

− установку защиты доступа для заданного (с клавиатуры) региона

памяти и ее проверку (функция Win32 API – VirtualProtect). 2. Запустите

приложение и проверьте его работоспособность на нескольких наборах

вводимых данных. Запротоколируйте результаты в отчет. Дайте свои

комментарии в отчете относительно выполнения функций Win32 API.

2. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

**Задание 2.2**. Использование проецируемых файлов для обмена

данными между процессами.

Указания к выполнению.

1. Создайте два консольных приложения с меню (каждая

выполняемая функция и/или операция должна быть доступна по

отдельному пункту меню), которые выполняют:

− приложение-писатель создает проецируемый файл (функции

Win32 API – CreateFile, CreateFileMapping), проецирует фрагмент

файла в память (функции Win32 API – MapViewOfFile,

UnmapViewOfFile), осуществляет ввод данных с клавиатуры и их

запись в спроецированный файл;

− приложение-читатель открывает проецируемый файл (функция

Win32 API – OpenFileMapping), проецирует фрагмент файла в

память (функции Win32 API – MapViewOfFile, UnmapViewOfFile),

считывает содержимое из спроецированного файла и отображает

на экран.

2. Запустите приложения и проверьте обмен данных между

процессами, удостоверьтесь в надлежащем выполнении задания.

Запротоколируйте результаты в отчет. Дайте свои комментарии в

отчете относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

## 2.1. получение информации о вычислительной системе (функция

## Win32 API – GetSystemInfo);

## 

## Рисунок 1: информации о вычислительной системе

## 2.2. определение статуса виртуальной памяти (функция Win32 API –

## GlobalMemoryStatus);

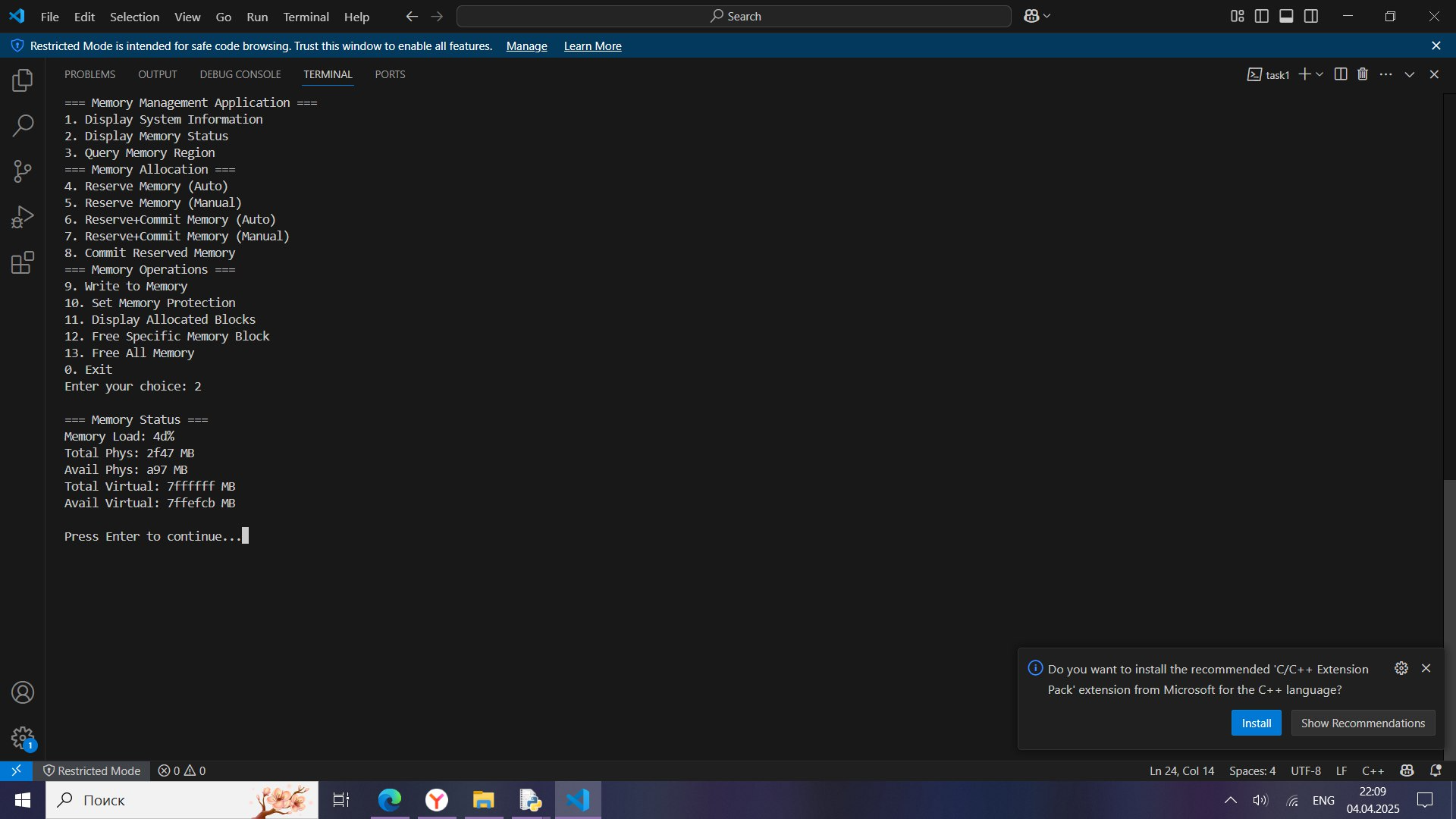


Рисунок 2: определение статуса виртуальной памяти

## 2.3. определение состояния конкретного участка памяти по заданному с клавиатуры адресу (функция Win32 API – VirtualQuery);

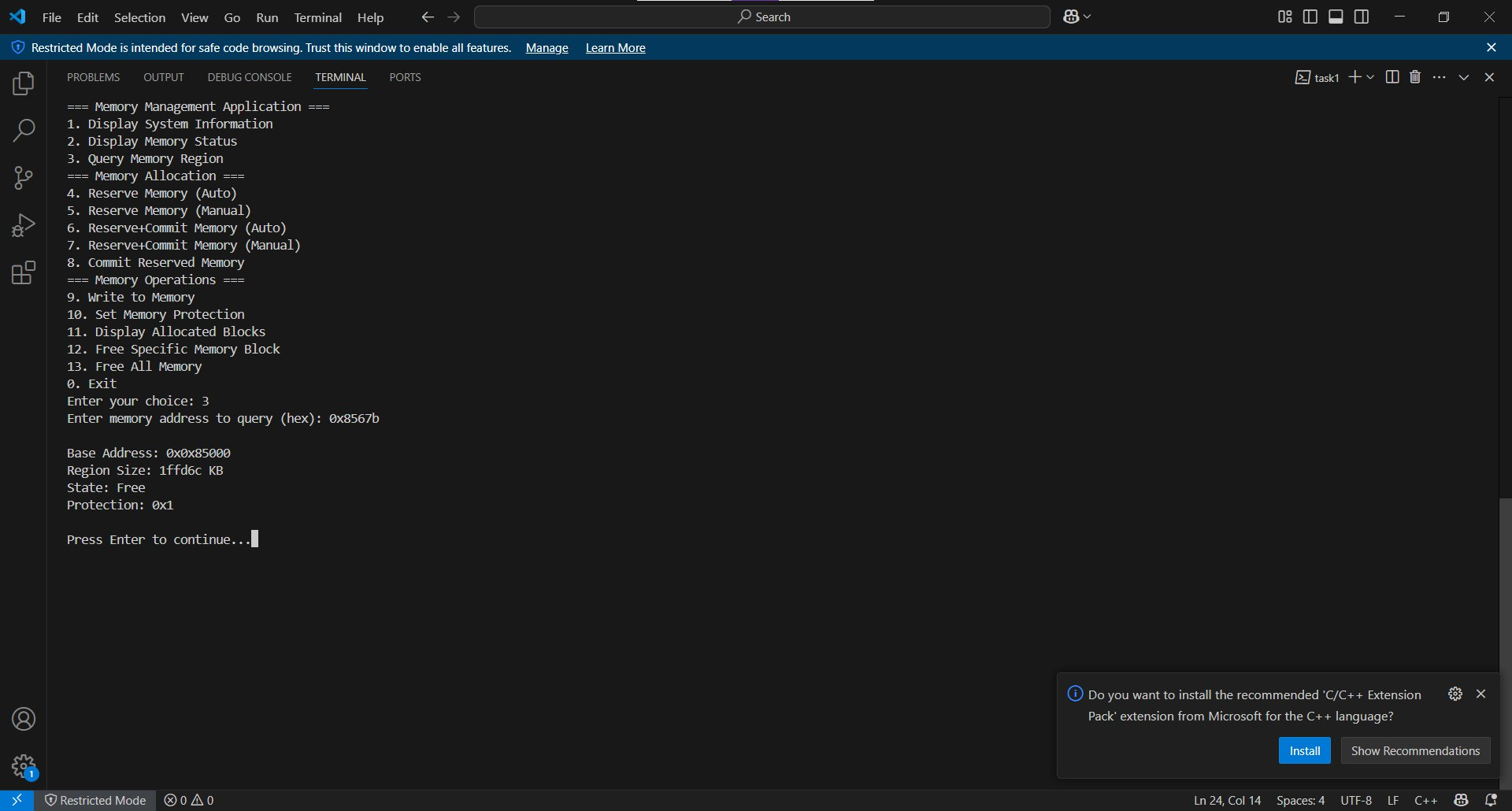
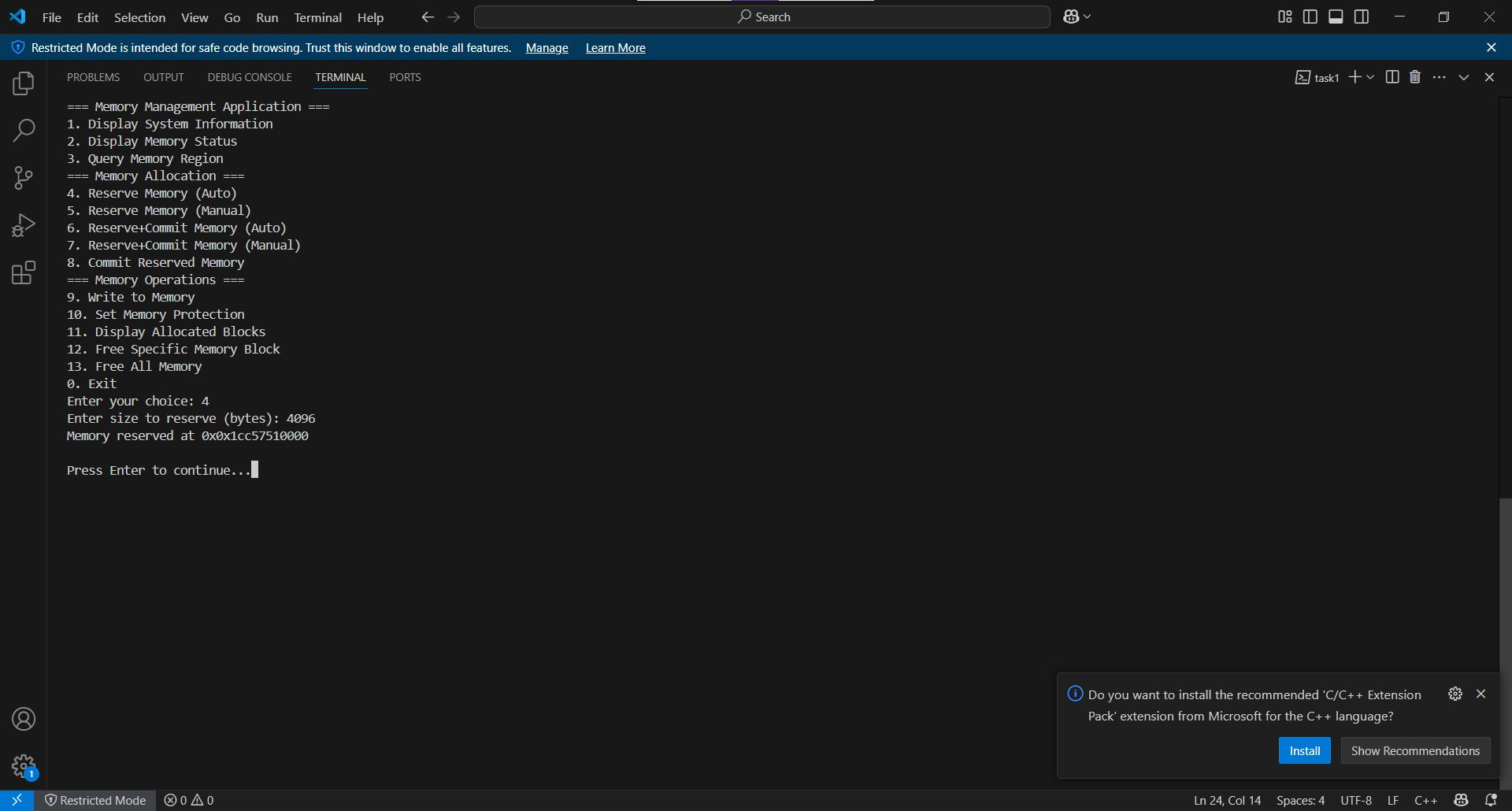
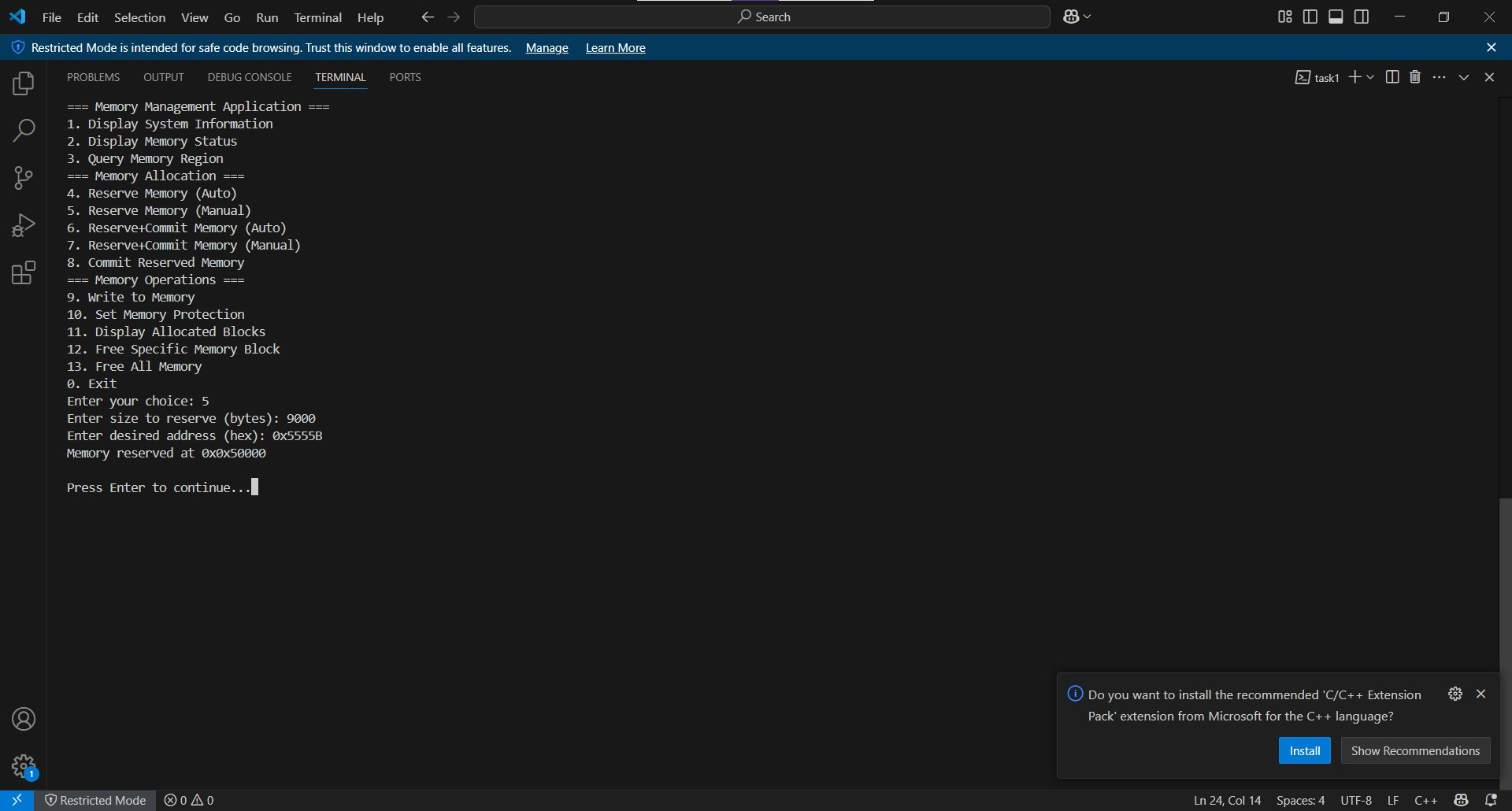


Рисунок 3: определение состояния конкретного участка памяти по заданному с клавиатуры адресу

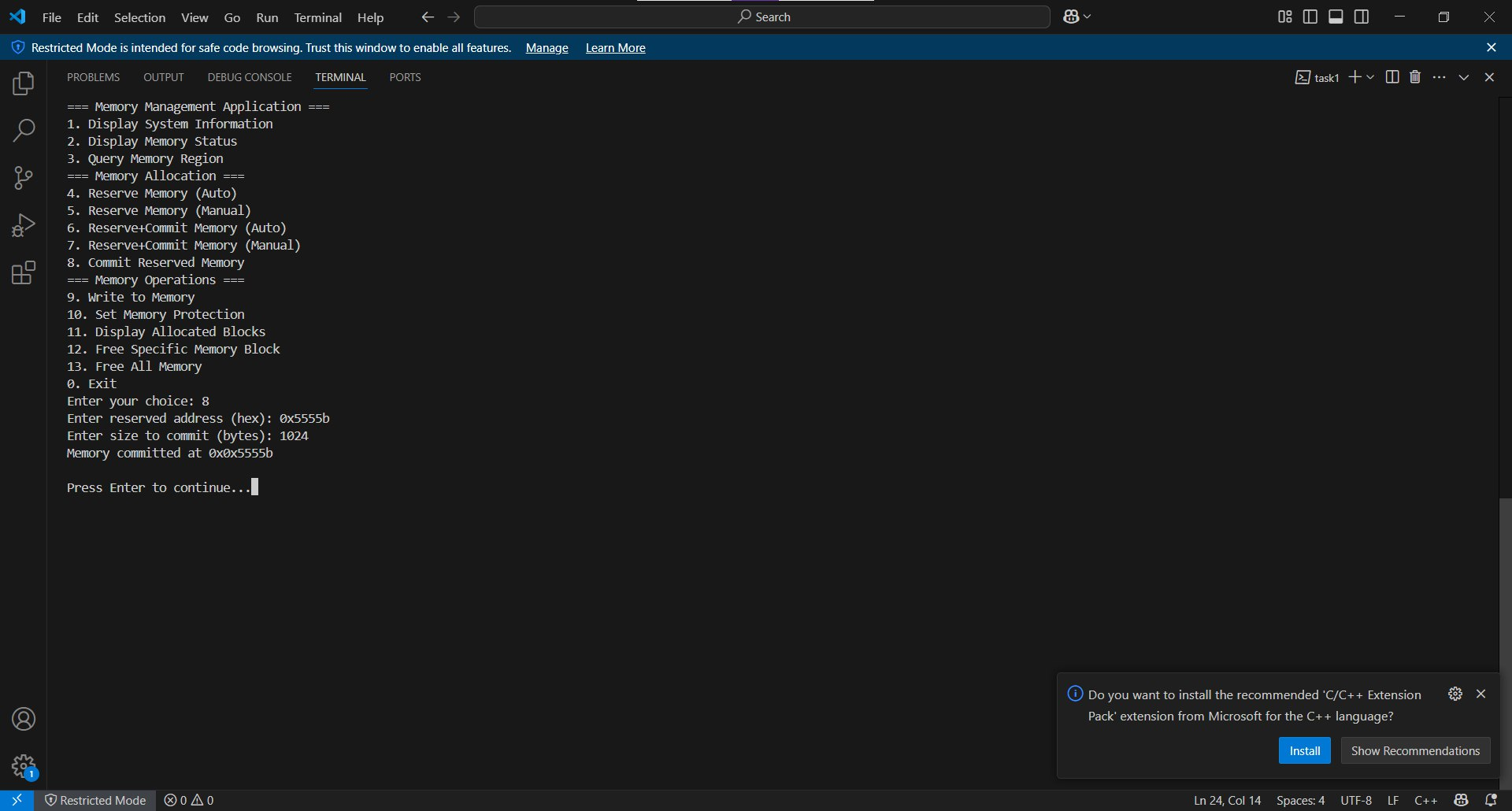
## 2.3. раздельное резервирование региона и передачу ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc, VirtualFree);





## 2.4. одновременное резервирование региона и передача ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc, VirtualFree);

## 

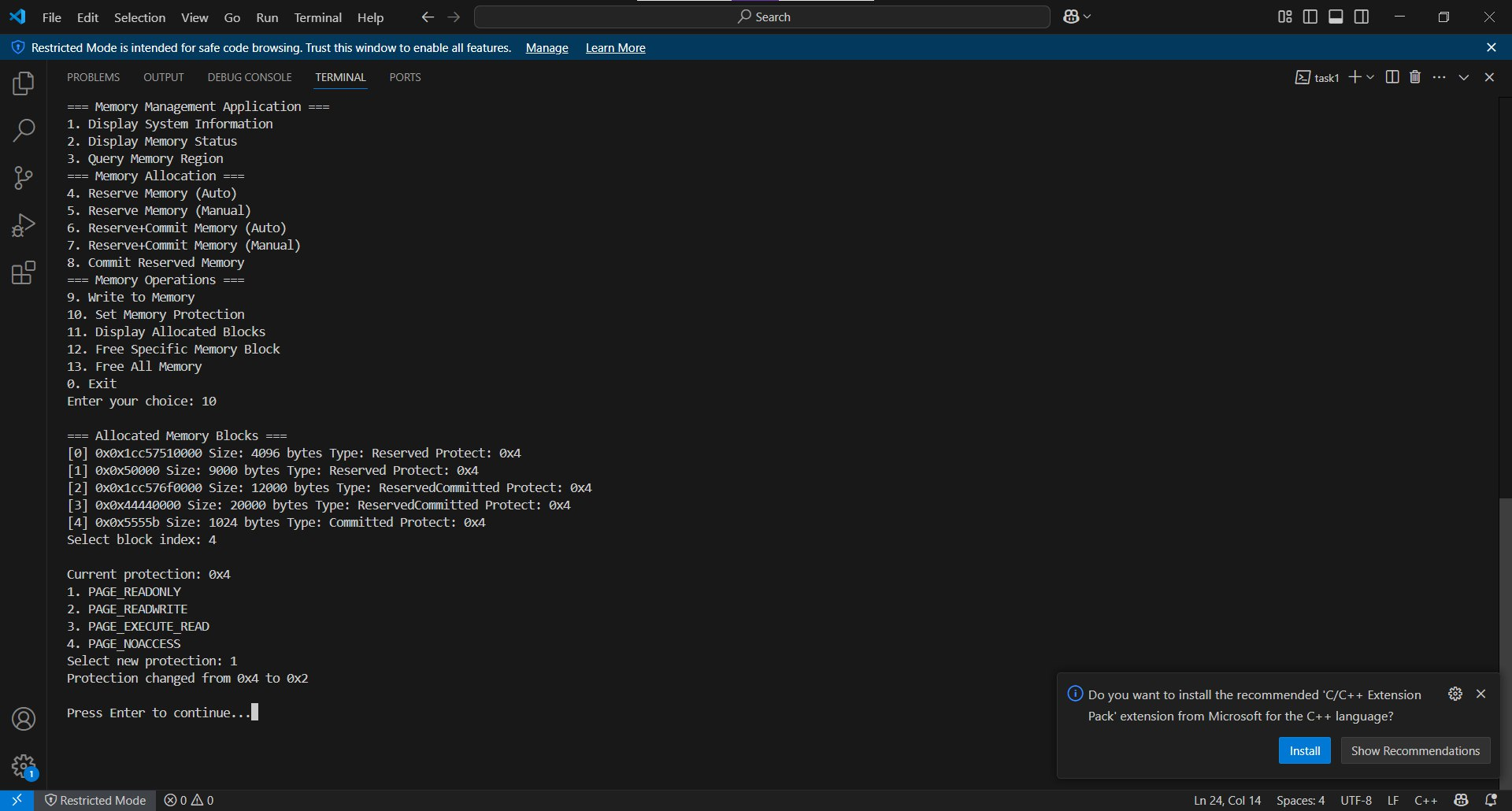


## 2.5. запись данных в ячейки памяти по заданным с клавиатуры

## Адресам;

## 

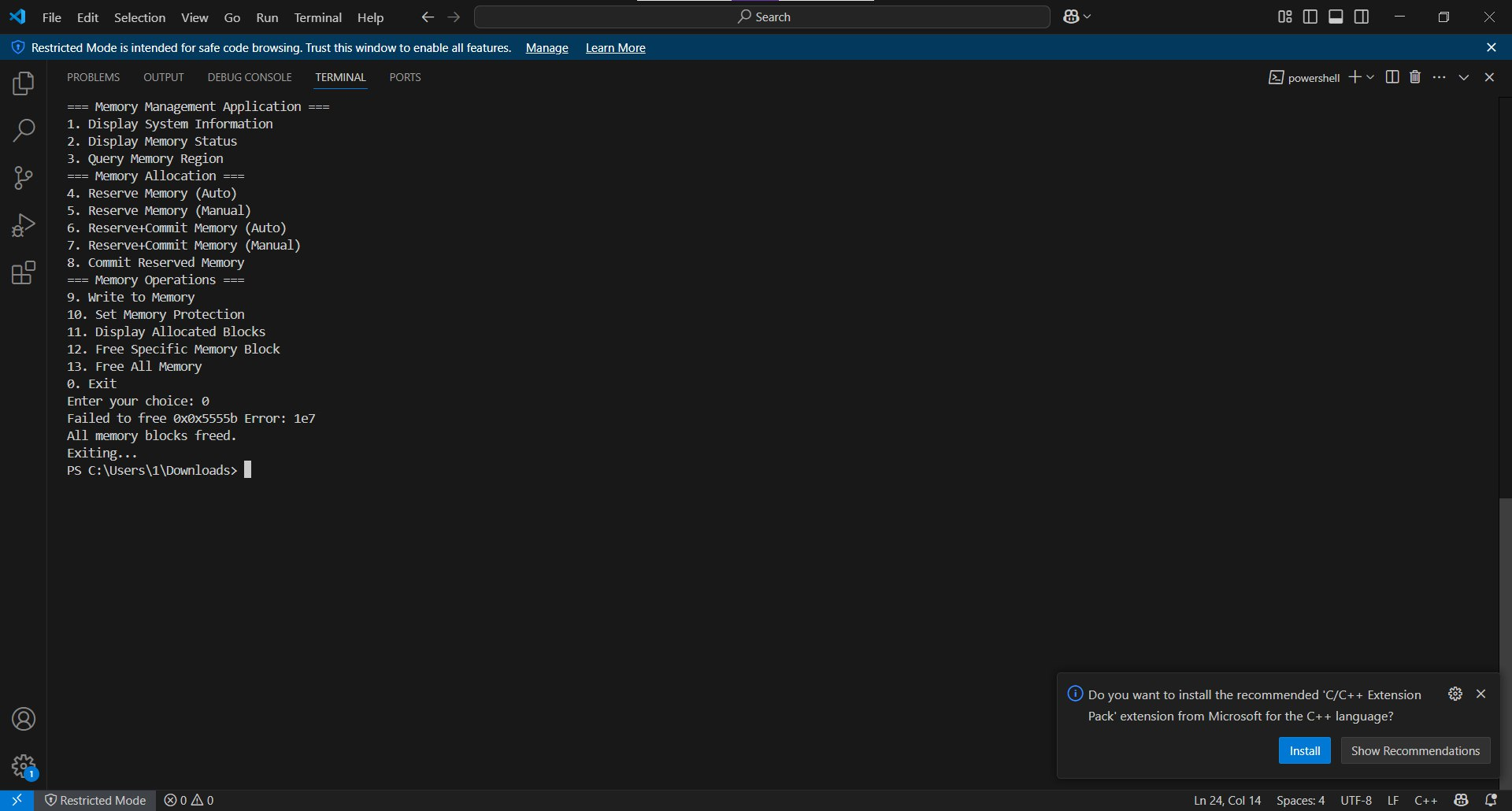
**2.6. установку защиты доступа для заданного (с клавиатуры) региона памяти и ее проверку (функция Win32 API – VirtualProtect).**



## 2.7. Допольнительных функции

## 

## 



## 2.7. Исходный код программы

**main.cpp**:

| #include <windows.h>  #include <iostream>  #include <iomanip>  #include <vector>  using namespace std;  struct MemoryBlock {  LPVOID address;  SIZE\_T size;  DWORD allocationType;  DWORD protection;  };  vector<MemoryBlock> allocatedBlocks;  // Prototypes  void displaySystemInfo();  void displayMemoryStatus();  void queryMemoryRegion();  void reserveMemory(bool manualAddress);  void reserveAndCommitMemory(bool manualAddress);  void commitReservedMemory();  void writeToMemory();  void setMemoryProtection();  void displayAllocatedBlocks();  void freeAllMemory();  void freeSpecificMemory();  int main() {  int choice;  do {  system("cls");  cout << "=== Memory Management Application ===" << endl;  cout << "1. Display System Information" << endl;  cout << "2. Display Memory Status" << endl;  cout << "3. Query Memory Region" << endl;  cout << "=== Memory Allocation ===" << endl;  cout << "4. Reserve Memory (Auto)" << endl;  cout << "5. Reserve Memory (Manual)" << endl;  cout << "6. Reserve+Commit Memory (Auto)" << endl;  cout << "7. Reserve+Commit Memory (Manual)" << endl;  cout << "8. Commit Reserved Memory" << endl;  cout << "=== Memory Operations ===" << endl;  cout << "9. Write to Memory" << endl;  cout << "10. Set Memory Protection" << endl;  cout << "11. Display Allocated Blocks" << endl;  cout << "12. Free Specific Memory Block" << endl;  cout << "13. Free All Memory" << endl;  cout << "0. Exit" << endl;  cout << "Enter your choice: ";  cin >> choice;  switch(choice) {  case 1: displaySystemInfo(); break;  case 2: displayMemoryStatus(); break;  case 3: queryMemoryRegion(); break;  case 4: reserveMemory(false); break;  case 5: reserveMemory(true); break;  case 6: reserveAndCommitMemory(false); break;  case 7: reserveAndCommitMemory(true); break;  case 8: commitReservedMemory(); break;  case 9: writeToMemory(); break;  case 10: setMemoryProtection(); break;  case 11: displayAllocatedBlocks(); break;  case 12: freeSpecificMemory(); break;  case 13: freeAllMemory(); break;  case 0: freeAllMemory(); cout << "Exiting..." << endl; break;  default: cout << "Invalid choice!" << endl;  }  if (choice != 0) {  cout << "\nPress Enter to continue...";  cin.ignore();  cin.get();  }  } while (choice != 0);  return 0;  }  void displaySystemInfo() {  SYSTEM\_INFO sysInfo;  GetSystemInfo(&sysInfo);  cout << "\n=== System Information ===" << endl;  cout << "Processor Architecture: ";  switch (sysInfo.wProcessorArchitecture) {  case PROCESSOR\_ARCHITECTURE\_AMD64: cout << "x64" << endl; break;  case PROCESSOR\_ARCHITECTURE\_INTEL: cout << "x86" << endl; break;  case PROCESSOR\_ARCHITECTURE\_ARM: cout << "ARM" << endl; break;  default: cout << "Unknown" << endl; break;  }  cout << "Page Size: " << sysInfo.dwPageSize << " bytes" << endl;  cout << "Allocation Granularity: " << sysInfo.dwAllocationGranularity << " bytes" << endl;  cout << "Min App Address: 0x" << hex << sysInfo.lpMinimumApplicationAddress << endl;  cout << "Max App Address: 0x" << hex << sysInfo.lpMaximumApplicationAddress << endl;  cout << "Number of Processors: " << sysInfo.dwNumberOfProcessors << endl;  cout << "Processor Type: " << sysInfo.dwProcessorType << endl;  }  void displayMemoryStatus() {  MEMORYSTATUSEX memStatus = { sizeof(MEMORYSTATUSEX) };  GlobalMemoryStatusEx(&memStatus);  cout << "\n=== Memory Status ===" << endl;  cout << "Memory Load: " << memStatus.dwMemoryLoad << "%" << endl;  cout << "Total Phys: " << memStatus.ullTotalPhys / (1024 \* 1024) << " MB" << endl;  cout << "Avail Phys: " << memStatus.ullAvailPhys / (1024 \* 1024) << " MB" << endl;  cout << "Total Virtual: " << memStatus.ullTotalVirtual / (1024 \* 1024) << " MB" << endl;  cout << "Avail Virtual: " << memStatus.ullAvailVirtual / (1024 \* 1024) << " MB" << endl;  }  void queryMemoryRegion() {  LPVOID address;  cout << "Enter memory address to query (hex): 0x";  cin >> hex >> address;  MEMORY\_BASIC\_INFORMATION memInfo;  if (VirtualQuery(address, &memInfo, sizeof(memInfo))) {  cout << "\nBase Address: 0x" << memInfo.BaseAddress << endl;  cout << "Region Size: " << memInfo.RegionSize / 1024 << " KB" << endl;  cout << "State: " << (memInfo.State == MEM\_COMMIT ? "Committed" :  memInfo.State == MEM\_RESERVE ? "Reserved" : "Free") << endl;  cout << "Protection: 0x" << hex << memInfo.Protect << endl;  } else {  cerr << "Query failed. Error: " << GetLastError() << endl;  }  }  void reserveMemory(bool manualAddress) {  SIZE\_T size;  LPVOID desiredAddress = NULL;  cout << "Enter size to reserve (bytes): ";  cin >> dec >> size;  if (manualAddress) {  cout << "Enter desired address (hex): 0x";  cin >> hex >> desiredAddress;  }  LPVOID address = VirtualAlloc(desiredAddress, size, MEM\_RESERVE, PAGE\_READWRITE);  if (address) {  allocatedBlocks.push\_back({address, size, MEM\_RESERVE, PAGE\_READWRITE});  cout << "Memory reserved at 0x" << hex << address << endl;  } else {  cerr << "Reservation failed. Error: " << GetLastError() << endl;  }  }  void reserveAndCommitMemory(bool manualAddress) {  SIZE\_T size;  LPVOID desiredAddress = NULL;  cout << "Enter size to allocate (bytes): ";  cin >> dec >> size;  if (manualAddress) {  cout << "Enter desired address (hex): 0x";  cin >> hex >> desiredAddress;  }  LPVOID address = VirtualAlloc(desiredAddress, size, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);  if (address) {  allocatedBlocks.push\_back({address, size, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE});  cout << "Memory allocated at 0x" << hex << address << endl;  } else {  cerr << "Allocation failed. Error: " << GetLastError() << endl;  }  }  void commitReservedMemory() {  LPVOID address;  SIZE\_T size;  cout << "Enter reserved address (hex): 0x";  cin >> hex >> address;  cout << "Enter size to commit (bytes): ";  cin >> dec >> size;  LPVOID result = VirtualAlloc(address, size, MEM\_COMMIT, PAGE\_READWRITE);  if (result) {  // Update existing block or add new?  bool found = false;  for (auto& block : allocatedBlocks) {  if (block.address == address) {  block.allocationType |= MEM\_COMMIT;  found = true;  break;  }  }  if (!found) {  allocatedBlocks.push\_back({address, size, MEM\_COMMIT, PAGE\_READWRITE});  }  cout << "Memory committed at 0x" << hex << address << endl;  } else {  cerr << "Commit failed. Error: " << GetLastError() << endl;  }  }  void writeToMemory() {  if (allocatedBlocks.empty()) {  cerr << "No allocated blocks available!" << endl;  return;  }  displayAllocatedBlocks();  size\_t index;  cout << "Select block index: ";  cin >> dec >> index;  if (index >= allocatedBlocks.size()) {  cerr << "Invalid index!" << endl;  return;  }  MemoryBlock& block = allocatedBlocks[index];  if (!(block.allocationType & MEM\_COMMIT)) {  cerr << "Memory not committed!" << endl;  return;  }  int value;  cout << "Enter integer value to write: ";  cin >> value;  // Simple write at start of block for demonstration  \*(int\*)block.address = value;  cout << "Value " << value << " written to 0x" << hex << block.address << endl;  // Verify  cout << "Read back value: " << dec << \*(int\*)block.address << endl;  }  void setMemoryProtection() {  if (allocatedBlocks.empty()) {  cerr << "No allocated blocks available!" << endl;  return;  }  displayAllocatedBlocks();  size\_t index;  cout << "Select block index: ";  cin >> dec >> index;  if (index >= allocatedBlocks.size()) {  cerr << "Invalid index!" << endl;  return;  }  MemoryBlock& block = allocatedBlocks[index];  cout << "\nCurrent protection: 0x" << hex << block.protection << endl;  cout << "1. PAGE\_READONLY\n2. PAGE\_READWRITE\n3. PAGE\_EXECUTE\_READ\n4. PAGE\_NOACCESS\n";  cout << "Select new protection: ";  int choice;  cin >> choice;  DWORD newProtect;  switch (choice) {  case 1: newProtect = PAGE\_READONLY; break;  case 2: newProtect = PAGE\_READWRITE; break;  case 3: newProtect = PAGE\_EXECUTE\_READ; break;  case 4: newProtect = PAGE\_NOACCESS; break;  default:  cerr << "Invalid choice!" << endl;  return;  }  DWORD oldProtect;  if (VirtualProtect(block.address, block.size, newProtect, &oldProtect)) {  block.protection = newProtect;  cout << "Protection changed from 0x" << hex << oldProtect  << " to 0x" << newProtect << endl;  } else {  cerr << "Failed to change protection. Error: " << GetLastError() << endl;  }  }  void displayAllocatedBlocks() {  if (allocatedBlocks.empty()) {  cout << "No memory blocks allocated." << endl;  return;  }  cout << "\n=== Allocated Memory Blocks ===" << endl;  for (size\_t i = 0; i < allocatedBlocks.size(); ++i) {  const auto& block = allocatedBlocks[i];  cout << "[" << i << "] 0x" << hex << block.address  << " Size: " << dec << block.size << " bytes"  << " Type: " << (block.allocationType & MEM\_RESERVE ? "Reserved" : "")  << (block.allocationType & MEM\_COMMIT ? "Committed" : "")  << " Protect: 0x" << hex << block.protection << endl;  }  }  void freeSpecificMemory() {  if (allocatedBlocks.empty()) {  cerr << "No allocated blocks available!" << endl;  return;  }  displayAllocatedBlocks();  size\_t index;  cout << "Select block index to free: ";  cin >> dec >> index;  if (index >= allocatedBlocks.size()) {  cerr << "Invalid index!" << endl;  return;  }  if (VirtualFree(allocatedBlocks[index].address, 0, MEM\_RELEASE)) {  cout << "Memory at 0x" << hex << allocatedBlocks[index].address << " freed." << endl;  allocatedBlocks.erase(allocatedBlocks.begin() + index);  } else {  cerr << "Free failed. Error: " << GetLastError() << endl;  }  }  void freeAllMemory() {  if (allocatedBlocks.empty()) {  cout << "No memory to free." << endl;  return;  }  for (const auto& block : allocatedBlocks) {  if (!VirtualFree(block.address, 0, MEM\_RELEASE)) {  cerr << "Failed to free 0x" << hex << block.address  << " Error: " << GetLastError() << endl;  }  }  allocatedBlocks.clear();  cout << "All memory blocks freed." << endl;  } |
| --- |

## 2.8. Выводы

В ходе данной работы было разработано консольное приложение на C++, предоставляющее удобный интерфейс для работы с виртуальной памятью Windows через Win32 API. Программа успешно реализует все основные функции управления памятью, включая резервирование, выделение, освобождение и изменение атрибутов защиты памяти.

# 3. Использование проецируемых файлов для обмена

# данными между процессами.

## 3.1. Создание и запуск консольного приложения

## \*SERVER SIDE

## \*CLIENT SIDE

## 

## 3.2. Проверка работоспособности приложения :

## \*Server side:

## 

## \*Client side:

## 

## 3.4. Исходный код программы

**server.c**

| #include <stdio.h>  #include <errno.h>  #include <stdlib.h>  #include <string.h>  #include <fcntl.h>  #include <unistd.h>  #include <sys/mman.h>  #include <sys/select.h>  #define RESULT\_ERROR (-1)  #define MAX\_SIZE 4096  int menu();  char\* projection(int \*fd);  void writter(char\* ptr);  void sync\_data(int fd);  void \_\_exit(char \*ptr, int fd, const char \*filename);  int main() {  int choice = 0;  char \*ptr = NULL;  int fd = -1;  char filename[20];  do {  choice = menu();  switch (choice) {  case 1:  ptr = projection(&fd);  break;  case 2:  if (ptr != NULL) {  writter(ptr);  } else {  printf("Please perform mapping first.\n");  }  break;  case 3:  if (ptr != NULL && fd != -1) {  \_\_exit(ptr, fd, filename);  }  choice = -1; // Quitter la boucle  break;  default:  printf("Undefined value! Exiting...\n");  choice = -1;  break;  }  } while (choice != -1);  return 0;  }  int menu() {  printf("===MENU=== \n");  printf("1-Perform mapping \n");  printf("2-Write data \n");  printf("3-Exit \n");  printf("Press a number to make a choice: ");  int choice;  scanf("%d", &choice);  getchar(); // Nettoyer le buffer après scanf  return choice;  }  char\* projection(int \*fd) {  char filename[20];  printf("Enter the filename: ");  fgets(filename, sizeof(filename), stdin);  filename[strcspn(filename, "\n")] = '\0'; // Supprimer le caractère '\n'  // Ouvrir le fichier  \*fd = open(filename, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);  if (\*fd == RESULT\_ERROR) {  perror("Error opening file");  exit(EXIT\_FAILURE);  }  // Définir la taille du fichier  if (ftruncate(\*fd, MAX\_SIZE) == RESULT\_ERROR) {  perror("Error setting file size");  close(\*fd);  exit(EXIT\_FAILURE);  }  // Mapper le fichier en mémoire  char \*ptr = mmap(NULL, MAX\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, \*fd, 0);  if (ptr == MAP\_FAILED) {  perror("Error mapping file");  close(\*fd);  exit(EXIT\_FAILURE);  }  return ptr;  }  void writter(char \*ptr) {  char data[MAX\_SIZE];  printf("Enter the data: \n");  fgets(data, sizeof(data), stdin);  data[strcspn(data, "\n")] = '\0'; // Supprimer le caractère '\n'  strcpy(ptr, data); // Copier les données dans la mémoire partagée  printf("Data has been written.\n");  }  void sync\_data(int fd) {  fd\_set read\_fds;  struct timeval timeout;  FD\_ZERO(&read\_fds);  FD\_SET(fd, &read\_fds);  timeout.tv\_sec = 5;  timeout.tv\_usec = 0;  printf("Waiting for data to be available....\n");  int ready = select(fd + 1, &read\_fds, NULL, NULL, &timeout);  if (ready == RESULT\_ERROR) {  perror("Error in select");  } else if (ready == 0) {  printf("Timeout: No data available.\n");  } else {  if (FD\_ISSET(fd, &read\_fds)) {  printf("Data is available for reading.\n");  }  }  }  void \_\_exit(char \*ptr, int fd, const char \*filename) {  if (munmap(ptr, MAX\_SIZE) == RESULT\_ERROR) {  perror("Error unmapping memory");  }  close(fd);  unlink(filename);  printf("Server has exited and cleaned up resources.\n");  } |
| --- |

Client.c

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/select.h>

#define RESULT\_ERROR (-1)

#define MAX\_SIZE 4096

int menu();

void read\_data();

void \_\_exit();

int main() {

int choice = 0;

do {

choice = menu();

switch (choice) {

case 1:

printf("Mapping not implemented separately in this example.\n");

break;

case 2:

read\_data();

break;

case 3:

printf("Exiting...\n");

choice = -1; // Quitter la boucle

break;

default:

printf("Undefined value! Exiting...\n");

choice = -1;

break;

}

} while (choice != -1);

return 0;

}

int menu() {

printf("===MENU=== \n");

printf("1-Perform mapping \n");

printf("2-Read data \n");

printf("3-Exit \n");

printf("Press a number to make a choice: ");

int choice;

scanf("%d", &choice);

getchar(); // Nettoyer le buffer après scanf

return choice;

}

void read\_data() {

char filename[20];

printf("Enter the filename: ");

fgets(filename, sizeof(filename), stdin);

filename[strcspn(filename, "\n")] = '\0'; // Supprimer le caractère '\n'

// Ouvrir le fichier

int fd = open(filename, O\_RDWR);

if (fd == RESULT\_ERROR) {

perror("Error opening file");

exit(EXIT\_FAILURE);

}

// Mapper le fichier en mémoire

char \*ptr = mmap(NULL, MAX\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (ptr == MAP\_FAILED) {

perror("Error mapping file");

close(fd);

exit(EXIT\_FAILURE);

}

// Attendre que les données soient disponibles (simplifié ici)

printf("Waiting for data to be available...\n");

sleep(2); // Simuler une attente (remplacez par un mécanisme de synchronisation réel)

// Lire et afficher les données

printf("Client received: %s\n", ptr);

// Nettoyer les ressources

munmap(ptr, MAX\_SIZE);

close(fd);

}

void \_\_exit() {

printf("Client has exited.\n");

exit(EXIT\_SUCCESS);

}

## 3.5. Выводы

В ходе выполнения работы я изучил механизм проецируемых файлов (memory-mapped files, MMAP) и его роль в организации взаимодействия процессов через общую память. Этот механизм позволяет отображать содержимое файла непосредственно в виртуальное адресное пространство процесса, что обеспечивает эффективный доступ к данным без необходимости использования стандартных операций чтения/записи.