Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Комсомольский-на-Амуре государственный университет»

Факультет компьютерных технологий

Кафедра «МОП ЭВМ»

ЛАБОРАТОРНАЯ РАБОТА 3

по дисциплине «Операционные системы и оболочки»

## Управление памятью в операционной системе Windows

Студент группы 6ИСб-1 О. Хусейнов

Преподаватель В.А Тихомиров

2018

**Цель работы**: Изучить теоретические вопросы управления памятью в ОС Windows и освоить приемы практической реализации этого управления с использованием системных функций.

**Задания:**

Выделить память под массив 1000х1000 элементов типа double и за-полнить его случайными числами в интервале от 1 до 10, защитить страницы памяти с массивом от записи, выдать сумму и среднее арифметическое элементов массива, дать команду на обнуление элементов и получить системное предупреждение о невозможности записи в массив.

**Выполнение:**

В наше время в основе многих компьютерных архитектур располагается специ­альный механизм управления памятью, который называют MMU (Memory Management Unit — устройство управления памятью).

Управлять работой MMU можно при помощи вызова VirtualAlloc. В качестве аргументов функция принимает адрес области памяти, ее размер, режим выделения и уровень защиты, Если в качестве первого аргумента функ­ции передать значение NULL, операционная система самостоятельно определит подходящий адрес выделяемой области. Вместо этого вы можете попросить ее выделить память с конкретным начальным адресом (этот адрес должен соответ­ствовать границе страницы). Второй аргумент — количество байт, которое необ­ходимо выделить. Windows автоматически округлит это количество таким обра­зом, чтобы оно равнялось числу, кратному размеру страницы. Если вы хотите просто выделить память, в качестве третьего аргумента необходимо указать зна­чение МЕМ\_СОММ1Т, Последний аргумент — уровень защиты, который будет соответ­ствовать выделяемой памяти (табл.1). Очевидно, что использовать некоторые из перечисленных здесь атрибутов (например, PAGE\_READONLY) не имеет смысла, так как память не инициализирована начальными значениями.

Программа, код которой приведен в листинге 1, получает в индивидуальное пользование относительно большой уча­сток памяти, при этом используются три разных метода. По умолчанию про­грамма работает с блоком памяти объемом 75 Мбайт. Это значение приемлемо для тестирования механизмов выделения/освобождения памяти.

Листинг 1 – Текст файла prog1.cpp

|  |
| --- |
| #include <windows.h>  #include <iostream>  char \*p=NULL;  #define MEGABYTE 1048576  #define MSIZE 75  using namespace std;  void main(int argc, char \*argv[])  {  char choice;  int msize=0;  setlocale(LC\_ALL, "Russian");  if (argc>1) msize=atoi(argv[1]);  if (!msize) msize=MSIZE;  do  {  int valid=1;  if (p)  {  cout<<"\nОсвобождение памяти\n";  VirtualFree(p,0,MEM\_RELEASE);  p=NULL;  cout<<"Завершено\n";  }  cout<<"1 - Выделить " <<msize<< "MB памяти\n";  cout<<"2 - Зарезервировать " <<msize<<"MB памяти\n";  cout<<"3 - Зарезервировать " <<msize<<"MB памяти и выделить в ней 1MB\n";  cout<<"4 - Выход\n";  cout<<"Ваш выбор: ";  cin>>choice;  switch (choice)  {  case '1':  cout<<"\nВыделение памяти\n";  p=(char \*)VirtualAlloc(NULL, msize\*MEGABYTE,  MEM\_COMMIT,PAGE\_READWRITE);  break;  case '2':  cout<<"\nРезервирование памяти\n";  p=(char \*)VirtualAlloc(NULL, msize\*MEGABYTE,  MEM\_RESERVE, PAGE\_READWRITE);  break;  case '3':  cout<<"\nРезервирование и частичное выделение памяти\n";  p=(char \*)VirtualAlloc(NULL, msize\*MEGABYTE,  MEM\_RESERVE, PAGE\_READWRITE);  cout<<"\nВыделение\n";  VirtualAlloc(p, MEGABYTE, MEM\_COMMIT, PAGE\_READWRITE);  break;  case '4':  exit(0);  default:  cout<<"\nЧего-чего?\n";  valid=0;  }  if (p&&valid)  cout<<"Блок памяти @ "<<hex<<(unsigned int)p<<"\n";  if (!p&&valid) cout<<"Ошибка выделения памяти!\n";  } while (1);  } |

Результат работы программы показан на рисунке 1.

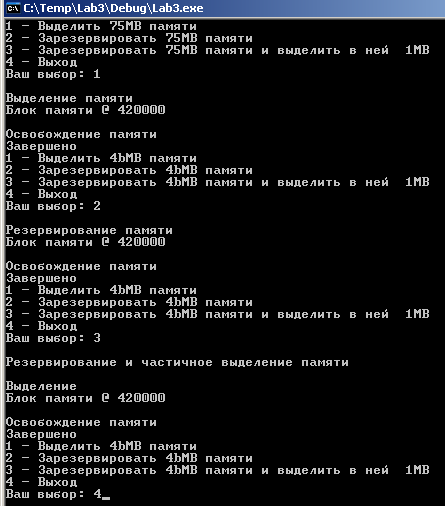


Рисунок 1 – Программа 1

Программа, код которой приведен в листинге 2, следит за возникновением исключений (он реагирует на любое возникающее исключение), таким образом, в случае некор­ректного обращения к защищенной странице программа сможет должным обра­зом отреагировать на это.

Листинг 2 – Текст файла prog2.cpp

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <windows.h>  #include <iostream>  char \*getmsg()  {  static char \*msg;  setlocale(LC\_ALL, "Russian");  if (msg==NULL)  {  char tmp[1024];  DWORD oldprot;  FILE \*f;  f=fopen("strings.txt", "r");  if (!f) return NULL;  fgets(tmp, sizeof(tmp), f);  // Выделяет целую страницу памяти, что приемлемо для данного примера  msg=(char \*)VirtualAlloc(NULL, strlen(tmp)+1, MEM\_COMMIT, PAGE\_READWRITE);  strcpy(msg, tmp);  fclose(f);  // Назначает уровень доступа к странице памяти только для чтения  VirtualProtect(msg, strlen(msg)+1, PAGE\_READONLY, &oldprot);  }  return msg;  }  void main()  {  char \*m=getmsg();  \_\_try  {  printf("Вся строка: %s\n",m);  m=strtok(m," \t");  printf("token1=%s\n",m);  m=getmsg();  printf("Вся строка: %s\n",m);  }  \_\_except (EXCEPTION\_EXECUTE\_HANDLER) //отлов системных исключений ОС Windows  {  printf("Произошло исключение!\n");  }  getchar(); // ожидание для отладочных целей  } |

В файле strings.txt записана строка «Hello world».

Результат работы программы показан на рисунке 2.

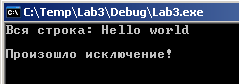


Рисунок 2 – Программа 2

Программа 3 имеет исходный код простой библиотеки, которая обеспечивает для основной программы простые средства работы с буфером. Код данной программы приведен в листинге 3.

Листинг 3 – Текст файла prog3.cpp

|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <exception>  #include <iostream>  static int maxindex;  char \*getbuffer()  {  SYSTEM\_INFO info;  // определение размера страницы  GetSystemInfo(&info);  char \*rv;  rv = (char \*)VirtualAlloc(NULL, maxindex = info.dwPageSize \* 1010,  MEM\_RESERVE, PAGE\_READWRITE);  return rv;  }  void cppexcept(unsigned int u, \_EXCEPTION\_POINTERS\* pExp)  {  throw u;  }  void putbuffer(char \*p, int index, char value)  {  \_\_try  {  p[index] = value;  }  \_\_except (EXCEPTION\_EXECUTE\_HANDLER)  {  std::cout << "Code = " << index << "\n";  if (EXCEPTION\_EXECUTE\_HANDLER != 0xC0000005)  {  printf("Неизвестное исключение\n");  system("pause");  exit(1);  }  // проверка: находится ли индекс в нужном диапазоне  if (index >= maxindex)  {  printf("Попытка доступа к ячейку за границами буфера\n");  system("pause");  exit(2);  }  // Выделяем один байт  // на самом деле система выделяет целую страницу  VirtualAlloc(p + index, 1, MEM\_COMMIT, PAGE\_READWRITE);  // еще одна попытка записать в буфер  putbuffer(p, index, value);  }  }  void freebuffer(char \*p)  {  VirtualFree(p, 0, MEM\_RELEASE);  }  //Основная программа (не может располагаться отдельно от библиотеки)  void main()  {  setlocale(LC\_ALL, "Russian");  char \*p = getbuffer();  for (int i = 0; i < 4000000; i++)  {  putbuffer(p, i, i & 0xFF);  }  freebuffer(p);  printf("Pass #2\n");  p = getbuffer();  // now backwards for variety  for (int j = 4000000 - 1; j >= 0; j--)  {  putbuffer(p, j, j & 0xFF);  }  freebuffer(p);  system("pause");  } |

Результат работы программы показан на рисунке 3.

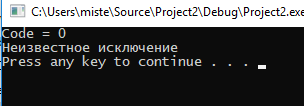


Рисунок 3 – Программа 3

Код программы, приведенной в задании указан в листинге 4.

Листинг 4 – Текста файла Source.cpp

|  |
| --- |
| /\*  Выделить память под массив 1000х1000 элементов типа double и заполнить его случайными числами в интервале от 1 до 10,  защитить страницы памяти с массивом от записи, выдать сумму и среднее арифметическое элементов массива,  дать команду на обнуление элементов и получить системное предупреждение о невозможности записи в массив.  \*/  #define \_CRT\_SECURE\_NO\_WARNINGS  // compile with: /EHa  #include <stdio.h>  #include <string.h>  #include <windows.h>  #include <locale.h>  #include <iostream>  using namespace std;  #define sizeMatrix 1000  int filter(unsigned int code, struct \_EXCEPTION\_POINTERS \*ep) {  if (code == EXCEPTION\_ACCESS\_VIOLATION) {  return EXCEPTION\_EXECUTE\_HANDLER;  }  else {  return EXCEPTION\_CONTINUE\_SEARCH;  };  }  int main()  {  setlocale(LC\_ALL, "rus");  system("color 0A ");  cout << "Выделение памяти под матрицу...\n";  // MEM\_COMMIT - Выполняется выделение страниц памяти для непосредственной работы с ними. Выделенные страницы заполняются нулями.  // PAGE\_READWRITE - Чтение и запись.  double\*matrix = (double\*)VirtualAlloc(NULL, sizeMatrix \* sizeMatrix \* sizeof(double), MEM\_COMMIT, PAGE\_READWRITE);  if (matrix == 0) {  MessageBox(0, "Память не выделена", "ERROR", MB\_ICONSTOP);  return 0;  }  else  {  cout << "Память под матрицу " << sizeMatrix << "\*" << sizeMatrix << " выделена\n";  }  cout << "Заполнение матрицы случайными числами...\n";  for (int i = 0; i < sizeMatrix; ++i)  for (int j = 0; j < sizeMatrix; ++j)  {  matrix[i \* sizeMatrix + j] = 1 + rand() % 10;  }  cout << "Матрица заполнена\n";  cout << "Установка защиты для памяти матрици...\n";  // Последний аргумент — это указатель на DWORD. Эта переменная будет содержать уровень безопасности, которым обла¬дал участок до обращения к VirtualProtect.  DWORD param;  VirtualProtect(matrix, sizeof(double), PAGE\_READONLY, &param);  cout << "Защита установлена\n";  cout << "Подсчет суммы всех элементов матрицы...\n";  long long sumMatrix = 0;  for (int i = 0; i < sizeMatrix; ++i)  for (int j = 0; j < sizeMatrix; ++j)  {  sumMatrix += matrix[i \* sizeMatrix + j];  }  cout << "Ответ: " << sumMatrix << "\n";  cout << "Подсчет среднего арифметического элементов матрицы...\n";  cout << "Ответ: " << (double)sumMatrix / (sizeMatrix\* sizeMatrix) << "\n";  cout << "Обнуление матрицы...\n";  \_\_try  {  for (int i = 0; i < sizeMatrix; ++i)  for (int j = 0; j < sizeMatrix; ++j)  {  matrix[i \* sizeMatrix + j] = 0;  }  cout << "Обнуление матрицы -> DONE" << endl;  }  \_\_except (filter(GetExceptionCode(), GetExceptionInformation()))  {  cout << "Обнуление матрицы -> ERROR" << endl;  }  system("pause");  return 0;  } |

Результат работы программы показан на рисунке 4.

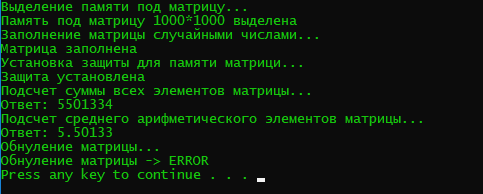


Рисунок 4 – Программа по индивидуальному заданию

**Список использованных источников**

1. Саймон, Р. Windows 2000 API Энцеклопедия программиста / Р. Саймон. - М: DiaSoft, 2002