**Задание 01. Windows**

1. Получите с помощью утилиты wmic информации об физической оперативной памяти компьютера, поясните эту информацию.

Результат представлен на рисунке 8.1

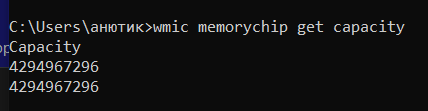


Рисунок 8.1

Команда wmic memorychip get capacity, предоставляет информацию об общей ёмкости модуля в байтах, таким образом память в гигабайтах = 4ГБ.

2. Получите с помощью утилиты powershell информации об физической оперативной памяти компьютера, поясните эту информацию. Результат представлен на рисунке 8.2

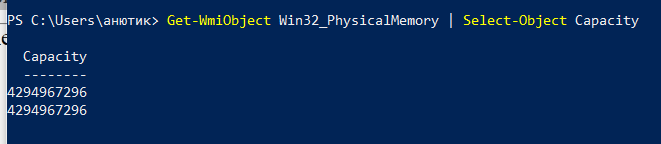


Рисунок 8.2

3. Получите с помощью утилиты systeminfo информации об оперативной памяти компьютера, поясните эту информацию. Результат представлен на рисунке 8.3

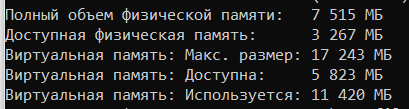


Рисунок 8.3

4. Получите с помощью утилиты performance monitor информации об оперативной памяти компьютера, поясните эту информацию.

**Задание 02.Windows**

5. Разработайте на языке программирования C# консольное приложение OS08\_02, которое каждые 5 секунд создает новый объект размером 128 МБ и выводит объем используемой памяти, установите Platform Target x86.

6. Понаблюдайте за выделенной памятью рабочим набором памяти приложения OS08\_02 в Диспетчере задач. Будет ли приложение работать бесконечно? Результат работы приложения представлен на рисунках 8.4 и 8.5

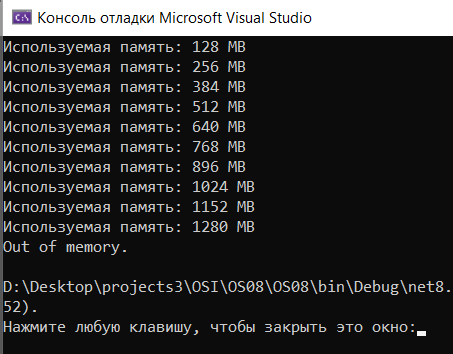


Рисунок 8.4

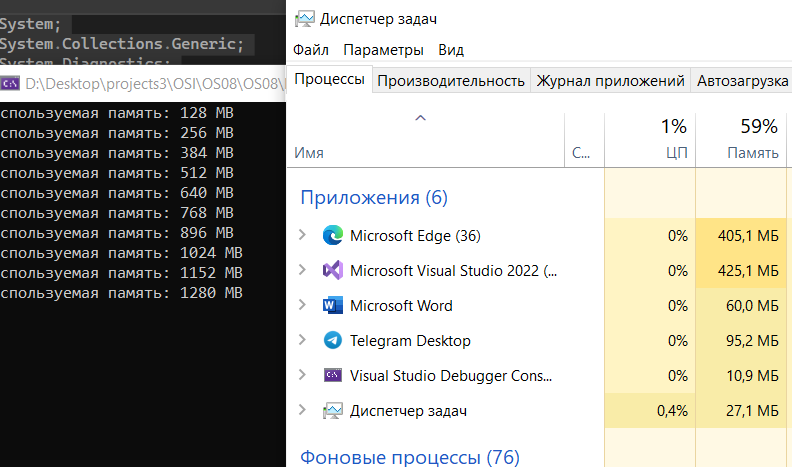


Рисунок 8.5

Нет, приложение не будет работать бесконечно. Приложение завершает свою работу, после того как займёт всю возможную память.

7. Добавьте в приложение OS08\_02 код, который заполняет произвольными значениями выделенную память, лучше всего в отдельном потоке. Изменилось ли соотношение выделенной памяти и рабочего набора?

Код приложения OS08\_02 представлен в таблице 8.1

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Threading;  using System.Threading.Tasks;  class Program  {  static void Main(string[] args)  {  List<Big> bigObjects = new List<Big>();  while (true)  {  Big bigObject = new Big();  bigObjects.Add(bigObject);  Task.Run(() => bigObject.FillArray());  long memoryUsed = GC.GetTotalMemory(true);  Console.WriteLine("Используемая память: {0} MB", (memoryUsed / (1024 \* 1024)));  Thread.Sleep(1000);  }  }  }  class Big  {  public Int32[] IntArray;  public Big()  {  IntArray = new int[128 \* 1024 \* 1024 / sizeof(int)];  }  public void FillArray()  {  Random rand = new Random();  for (int i = 0; i < IntArray.Length; i++)  {  IntArray[i] = rand.Next();  }  }  } |

Таблица 8.1

Результат работы приложения OS08\_02 представлен на рисунке 8.6

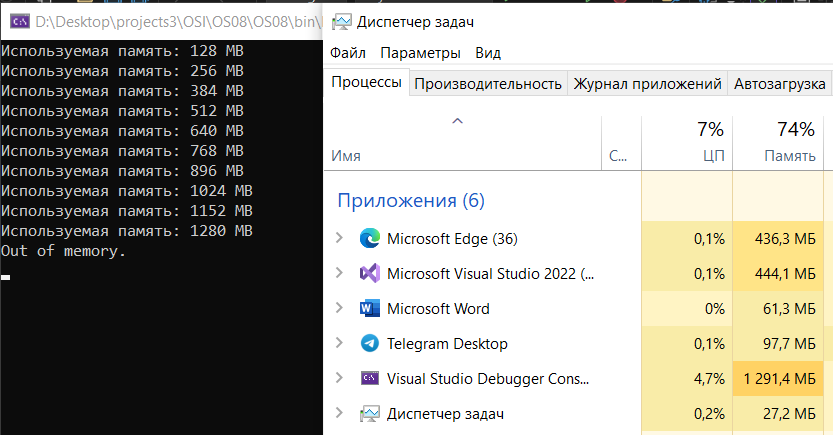


Рисунок 8.6

Когда мы запускаем приложение OS08\_02, с записанными в массивы данными, то наш процесс начинает занимать намного больше оперативной памяти и сильнее нагружать процессор.

8. Запустите несколько экземпляров приложения OS08\_02. Сравните частный и общий рабочие наборы. Результат представлен на рисунке 8.7

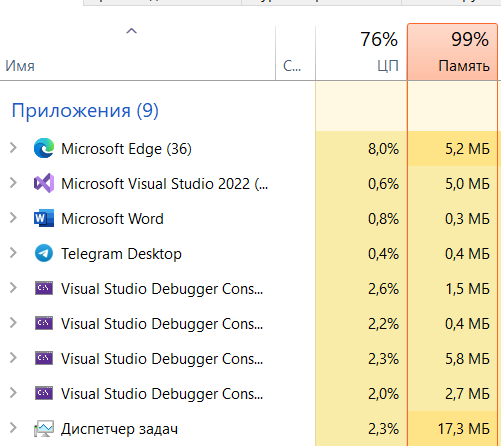


Рисунок 8.7

Чем больше экземпляров программы запускается, тем большая идёт нагрузка на процессор и система зависает.

**Задание 03.Linux**

9. Разработайте консольное приложение OS08\_03, выполняющее длинный цикл. Код программы представлен в таблице 8.2, а результат выполнения на рисунке 8.8 соответственно.

|  |
| --- |
| int main()  {  pid\_t pid = getpid();  nice(0);  for (int i = 0; i < 10000000; i++)  {  printf("[OS08\_06] PID = %d\n", pid);  sleep(1);  }  exit(0);  } |

Таблица 8.2

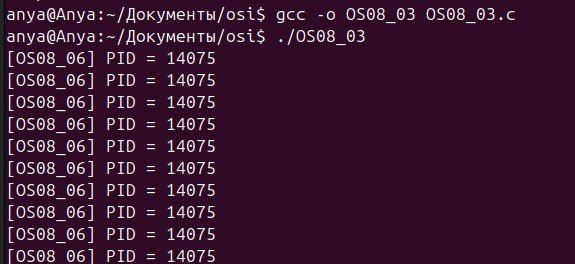


Рисунок 8.8

10. Продемонстрируйте с помощью файловой системы /proc структуру адресного пространства. Результат представлен на рисунке 8.9

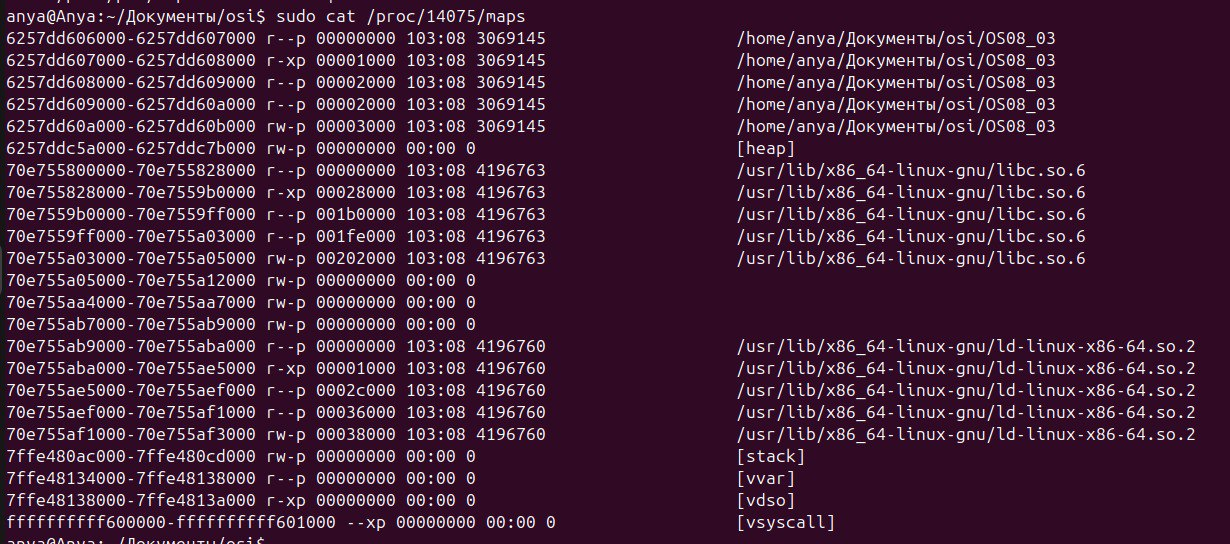


Рисунок 8.9

11. Продемонстрируйте с помощью pmap структуру адресного пространства. Результат представлен на рисунке 8.10

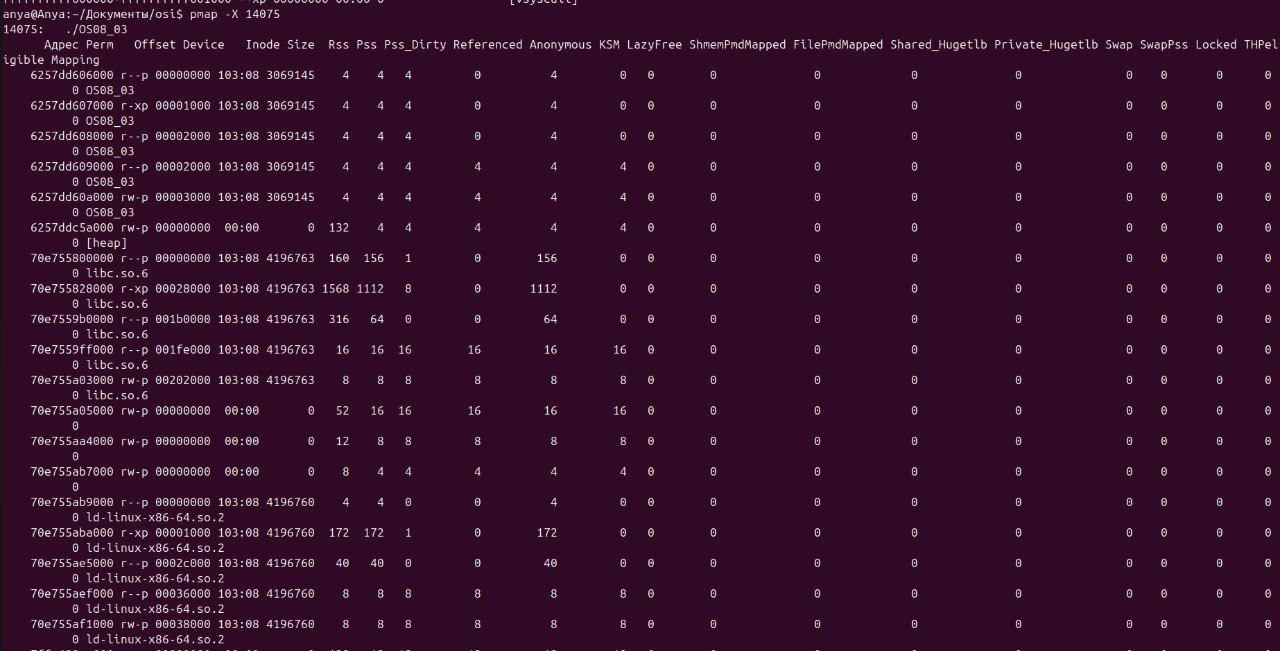


Рисунок 8.10

12. Определите с помощью утилиты objdump адрес загрузки mainмодуля, секций с кодом, данными, неинициализированными глобальными переменными.

Адрес загрузки mainмодуля представлен на рисунке 8.11

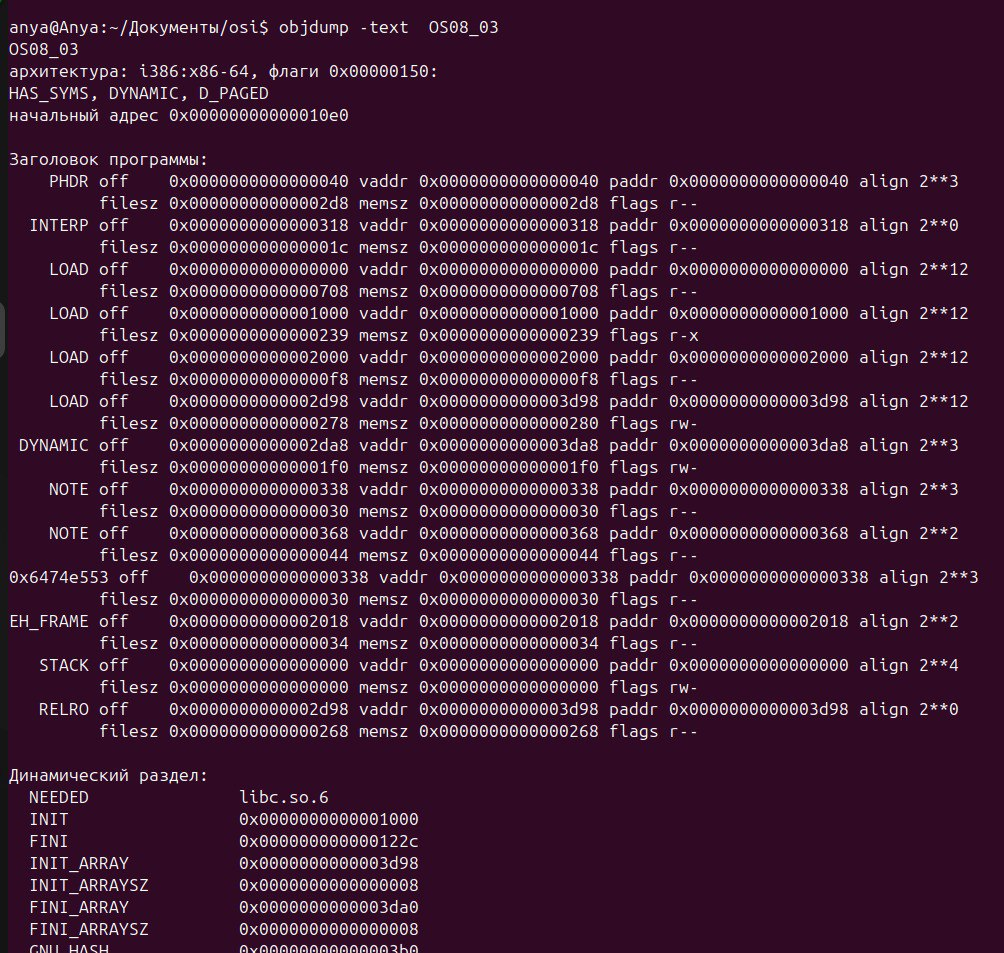


Рисунок 8.11

Адрес секции с данными представлен на рисунке 8.12

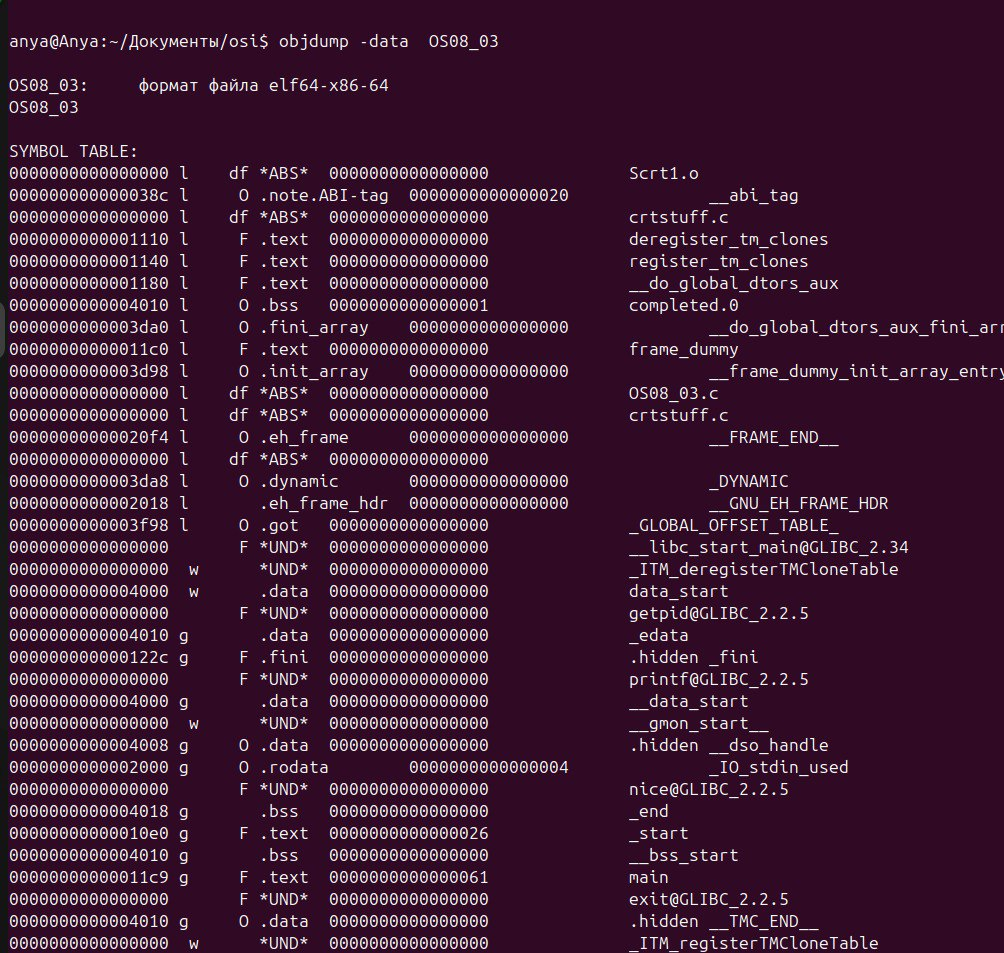


Рисунок 8.12

**Задание 04.**

Вариант 3. Укажите содержимое байта по виртуальному адресу 0x001700A5, если регистр CR3 содержит значение 0x13907000.

0х означает что адреса нам представлены в шестнадцатиричной системе исчисления и мы по-быстрому всё переводим в двоичную:

0x001700A5 - 0000 0000 0001 0111 0000 0000 1010 0101

0x13907000 - 0001 0011 1001 0000 0111 0000 0000 0000 - PGD

Первое задание сделано:

**Запишите виртуальный адрес в двоичном виде:** 0000 0000 0001 0111 0000 0000 1010 0101

Дальше мы разбиваем наш адрес на 3 секции

Старшие 10 бит – индекс каталога страниц - PDI

Средние 10 бит – индекс таблицы страниц - PTI

Младшие 12 бит – смещение по странице

**Индекс каталога страниц:** 0000 0000 00 – 0х0

**Индекс таблицы страниц:** 01 0111 0000 – 0х170

Смещение по странице: 0000 1010 0101 – 0хА5

CR3 содержит физический адрес самого верхнего уровня структуры страничной организации, называемого Page Global Directory (PGD).

Далле, с помощью смещения мы будем считать искомый физический адрес из адреса корневой таблицы страниц, то есть CR3.

Базовый адрес физической памяти страницы, мы смотрим по смещению с индексом каталога страниц:

Физический адрес записи в каталоге страниц = адрес корневой таблицы страниц + индекс каталога страниц \* 4(размер одной записи)

0x13907000 + 0х0 \* 4 = 0x13907000 – далее ищем запись в нашем дампе памяти, по этому адресу

В зависимости от используемых функций процессора в регистре CR3 наряду с адресом PGD может храниться и дополнительная информация, поэтому более универсальный способ получения физического адреса PGD из регистра CR3 заключается в маскировании нижних 12 битов его содержимого.



Следовательно дальше мы бу дем использовать адрес 0х13909127, так как нет смещения (последние 3 байта)

По такому же принципу как вычислялся адрес таблицы страниц, вычисляем и адрес страницы:

Физический адрес страницы = Физический адрес таблицы страниц + (Индекс таблицы страниц \* размер страницы)

0х13909000 + 0х170\*4 = 0х13909000 + 0х05С0 = 0х139095С0



**Базовый адрес физической страницы памяти**: 0x2600300

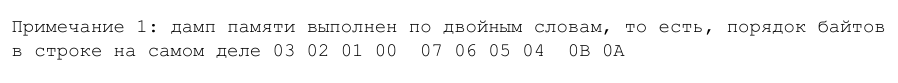
Искомый адрес мы находим по адресу страницы с использованием смещения по этой странице: 0x26003000 + 0хА5

Искомый физический адрес: 0x260030А5

Так как, нам представлены диапазоны адресов и адреса 0x260030А5 в пуле нету, то смотрим диапазон, в который он будет входить, то есть диапазон начинающийся с 0x260030А0. Такой диапазон в дампе присутствует. Рассмотрим его:  


Данный адрес предоставляет нам строку, содержащую 8 двухбайтовых слов с обратным (сначала младшие биты – потом старшие) порядком байтов.

По примеру:



Узнаем что на месте 5, которое указывает на смещение, у нас находятся данные F0

Требуемые данные: 0xF0