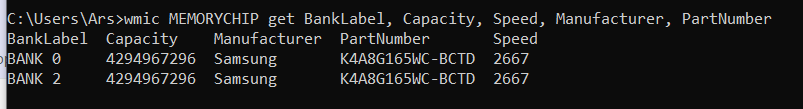
Операционные Системы

Лабораторная работа №8

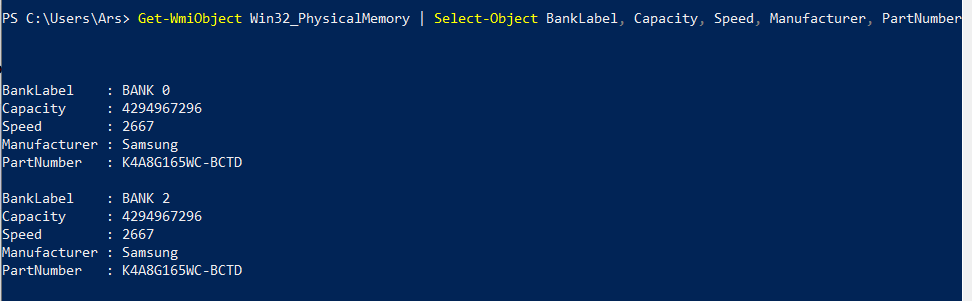
Авдей А.Ю 3курс 2-1

Задание 01. Windows

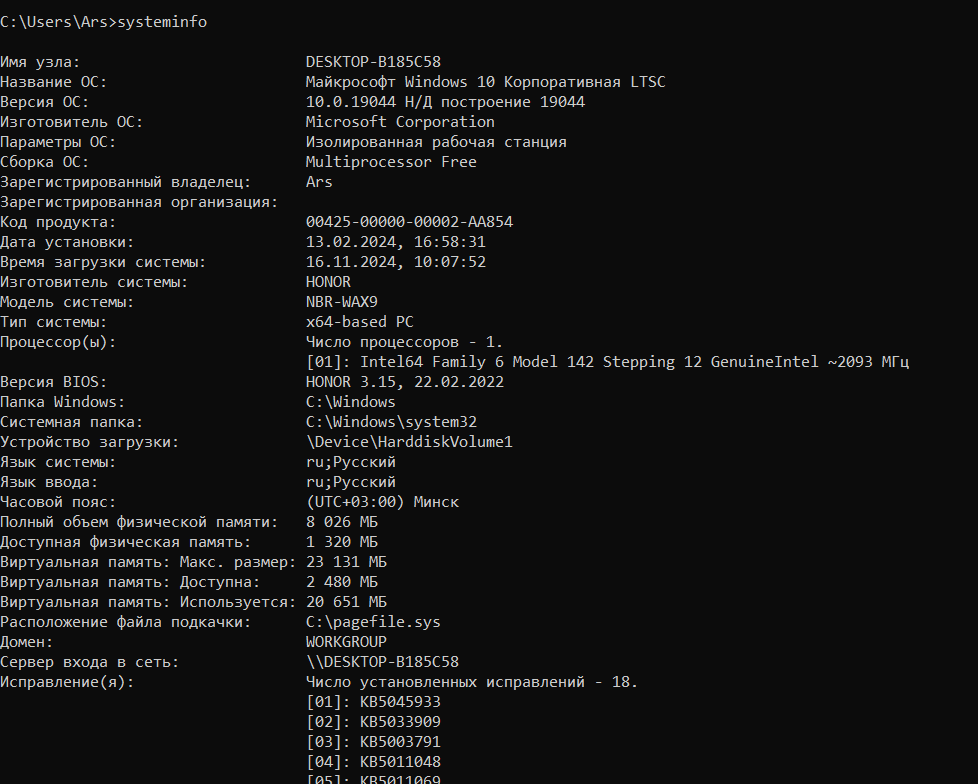
1. Получите с помощью утилиты wmic информации об физической оперативной памяти компьютера, поясните эту информацию.



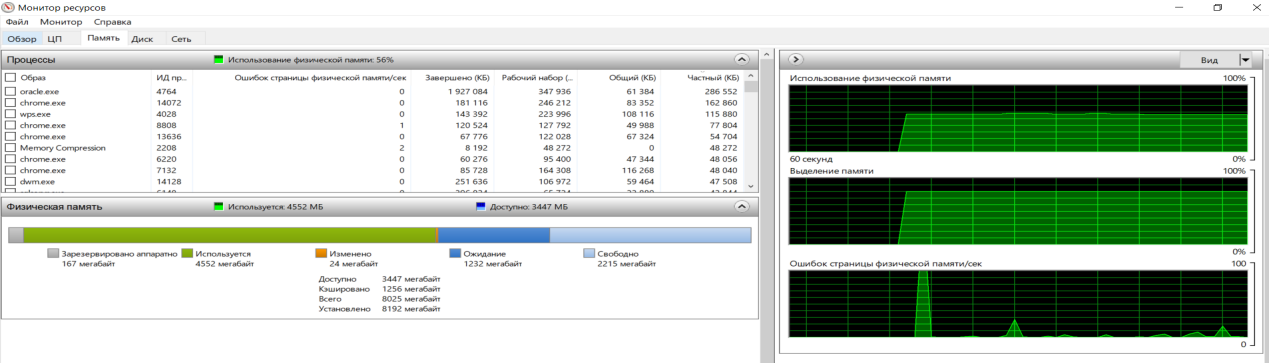
1. Получите с помощью утилиты powershell информации об физической оперативной памяти компьютера, поясните эту информацию.



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



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



Задание 02.Windows

1. 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.

using System;

using System.Diagnostics;

using System.Threading;

class Program

{

static void Main(string[] args)

{

// Размер объекта в 128 МБ

const int objectSizeInMB = 128;

byte[] largeObject = null;

long totalAllocatedMemory = 0;

while (true)

{

// Создание нового объекта размером 128 МБ

largeObject = new byte[objectSizeInMB \* 1024 \* 1024];

totalAllocatedMemory += objectSizeInMB; // Увеличиваем на 128 МБ

long usedMemory = Process.GetCurrentProcess().WorkingSet64;

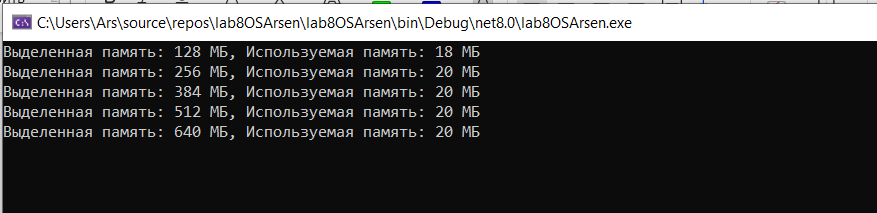
Console.WriteLine($"Выделенная память: {totalAllocatedMemory} МБ, Используемая память: {usedMemory / (1024 \* 1024)} МБ");

Thread.Sleep(5000);

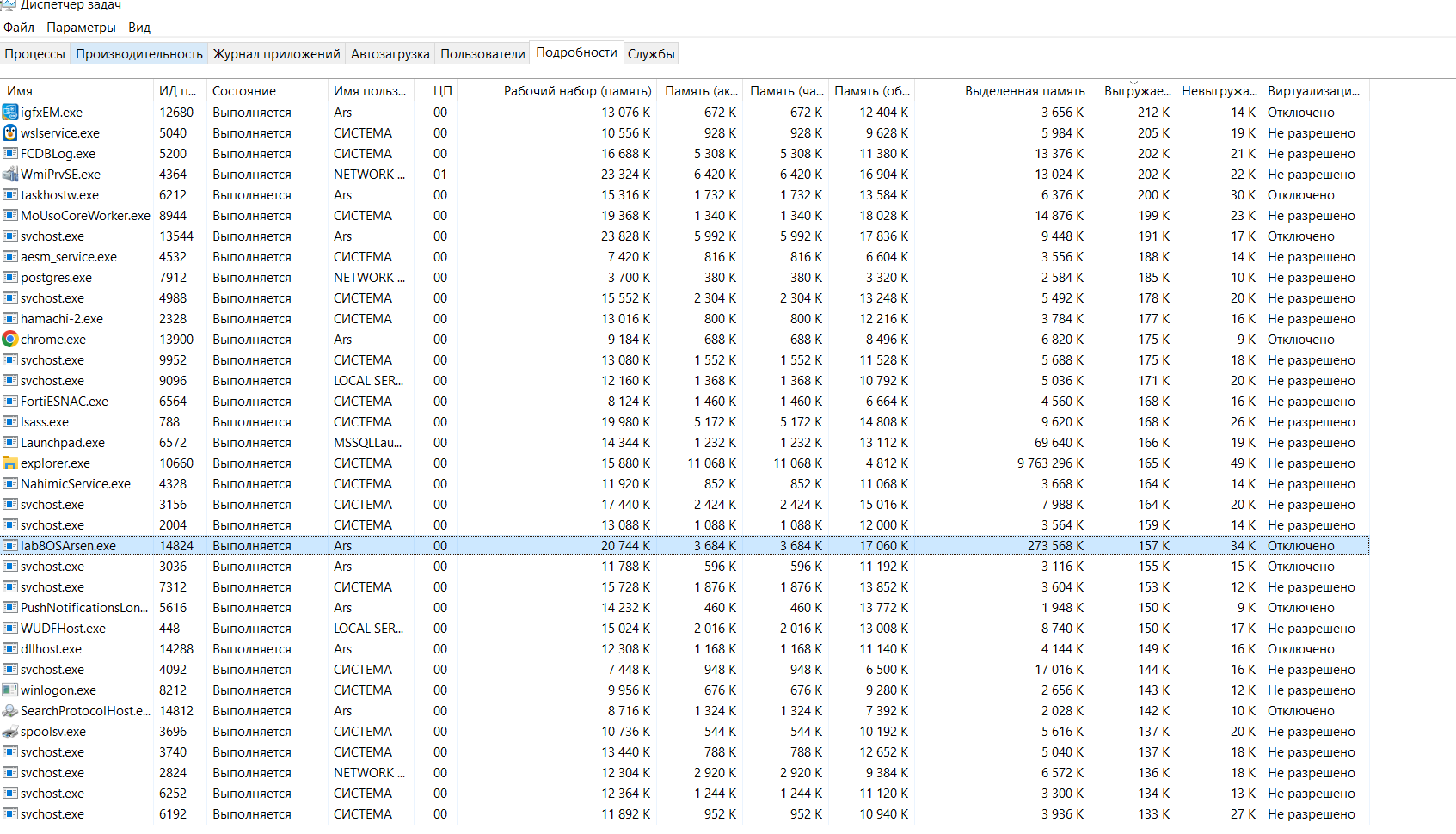
}

}

}



1. Понаблюдайте за выделенной памятью рабочим набором памяти приложения OS08\_02 в Диспетчере задач. Будет ли приложение работать бесконечно?



1. 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?

using System;

using System.Diagnostics;

using System.Threading;

using System.Threading.Tasks;

class Program

{

static void Main(string[] args)

{

const int objectSizeInMB = 128;

byte[] largeObject = null;

long totalAllocatedMemory = 0;

while (true)

{

largeObject = new byte[objectSizeInMB \* 1024 \* 1024];

totalAllocatedMemory += objectSizeInMB; // Увеличиваем на 128 МБ

// Запуск потока для заполнения памяти произвольными значениями

Task.Run(() => FillMemory(largeObject));

// Получение объема используемой памяти

long usedMemory = Process.GetCurrentProcess().WorkingSet64;

Console.WriteLine($"Выделенная память: {totalAllocatedMemory} МБ, Используемая память: {usedMemory / (1024 \* 1024)} МБ");

Thread.Sleep(5000);

}

}

static void FillMemory(byte[] array)

{

Random random = new Random();

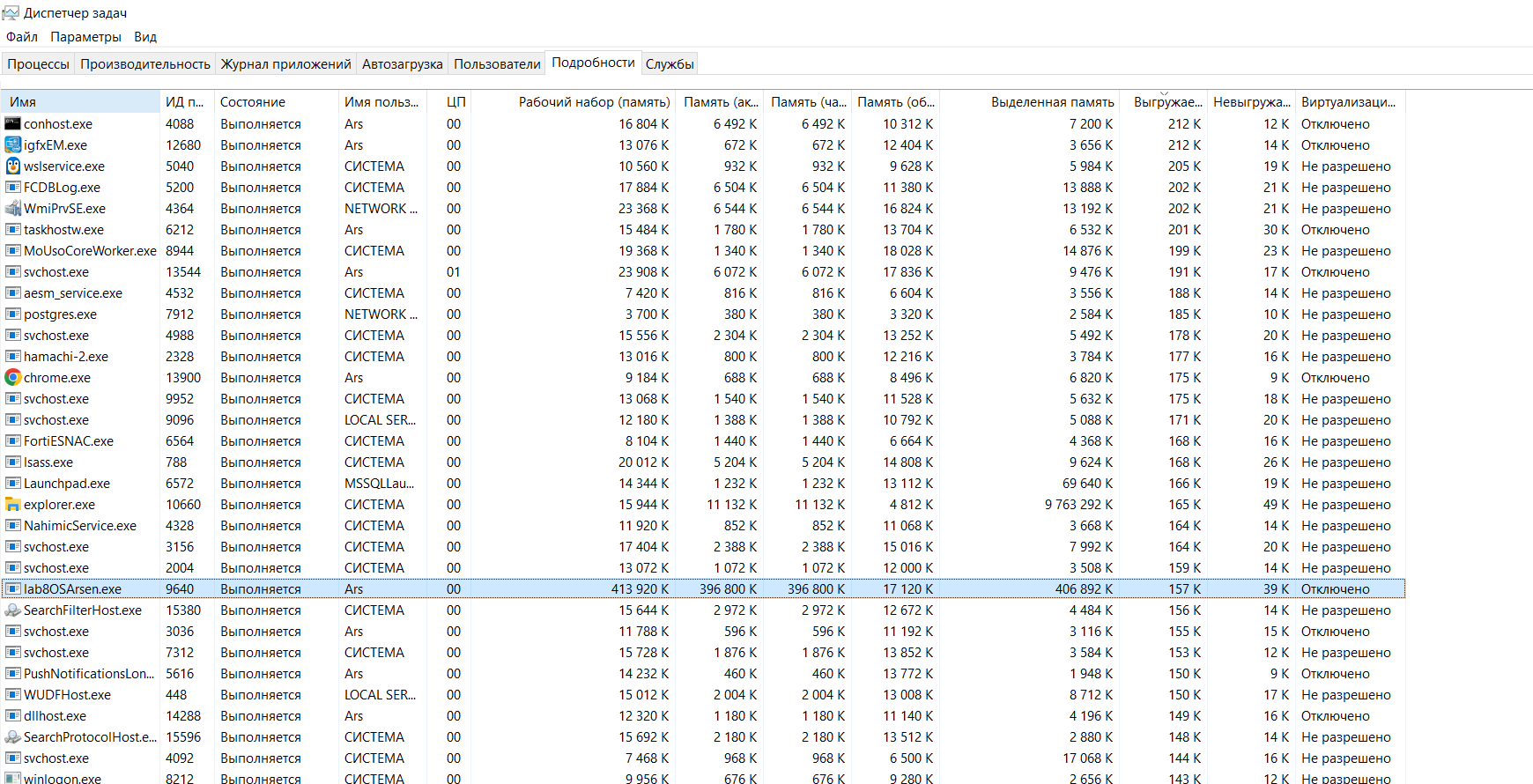
for (int i = 0; i < array.Length; i++)

{

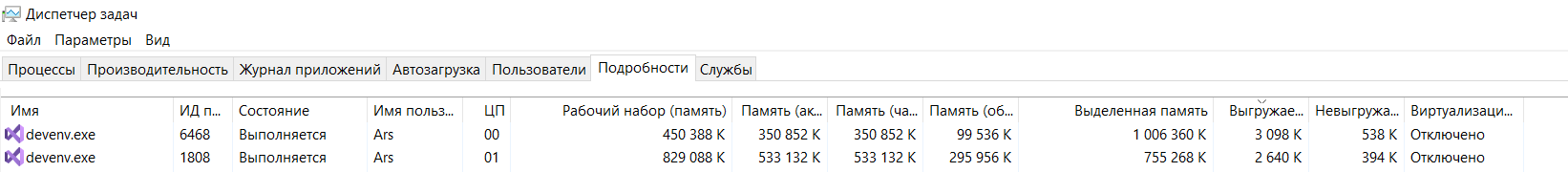
array[i] = (byte)random.Next(256); }

}

}



1. Запустите несколько экземпляров приложения OS08\_02.Сравните частный и общий рабочие наборы.



Задание 03.Linux

1. Разработайте консольное приложение OS08\_03, выполняющее длинный цикл.

root@DESKTOP-B185C58:~# nano os8ars.cpp

root@DESKTOP-B185C58:~# g++ -o os8ars os8ars.cpp

root@DESKTOP-B185C58:~# ./os8ars

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

while (1) {

printf("Работаю...\n");

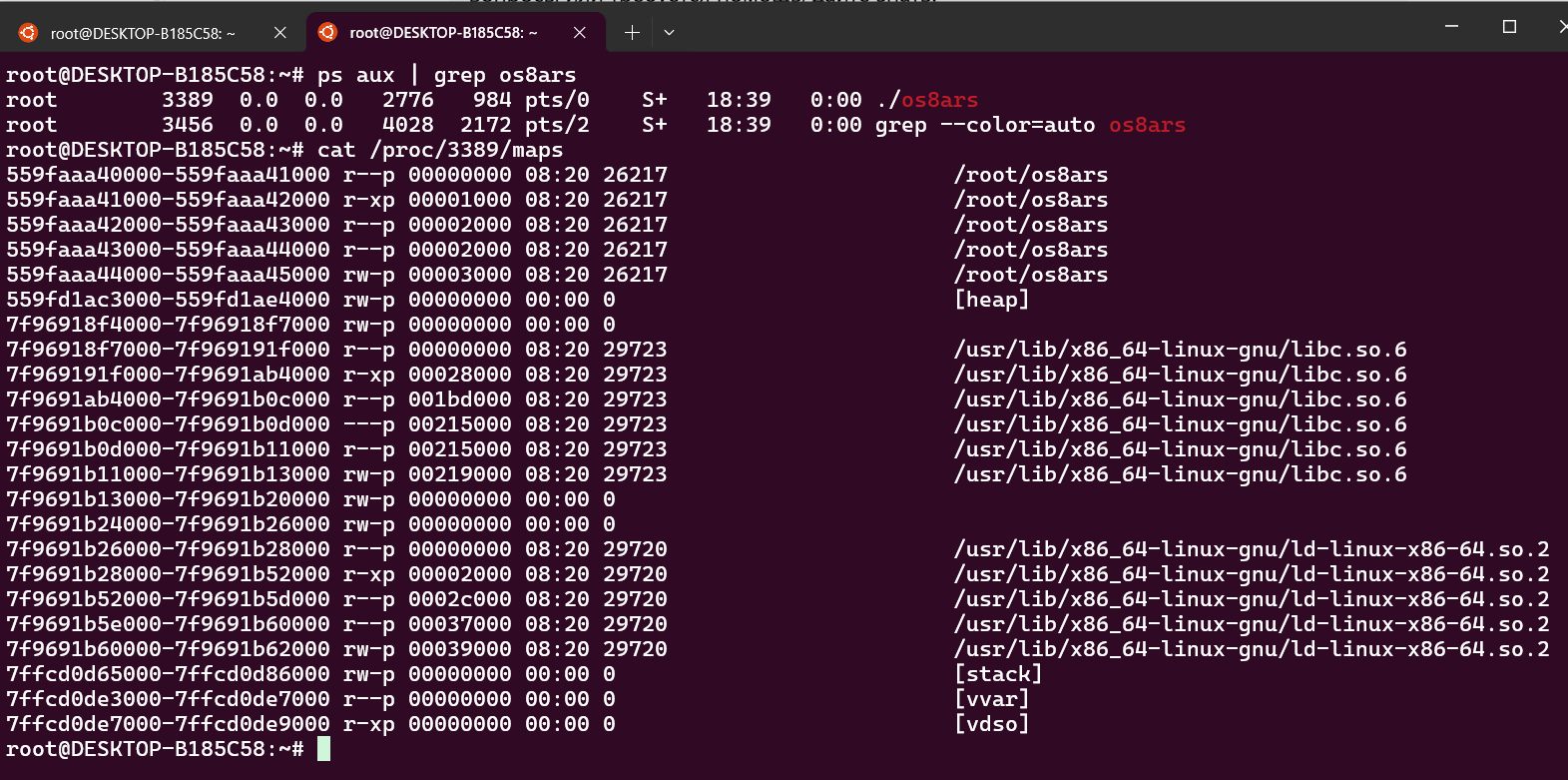
sleep(1);

}

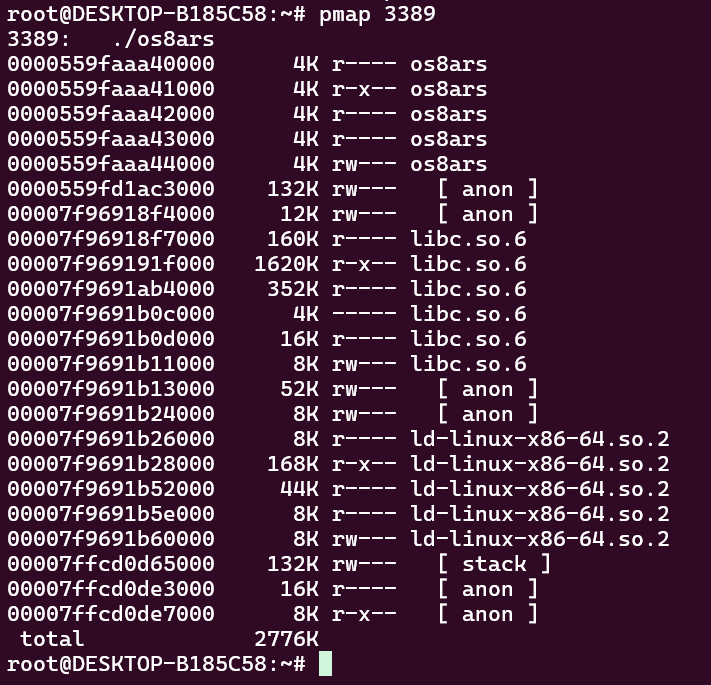
return 0;

}

1. Продемонстрируйте с помощью файловой системы /proc структуру адресного пространства.



1. Продемонстрируйте с помощью pmap структуру адресного пространства.

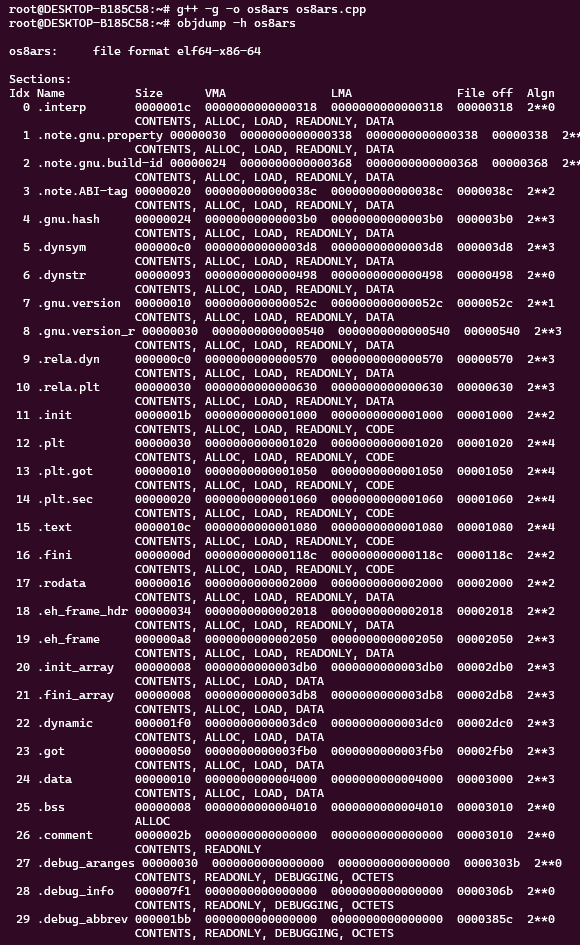


1. Определите с помощью утилиты objdump адрес загрузки main модуля, секций с кодом, данными, неинициализированными глобальными переменными.

.text: секция с кодом/main

.data: секция с инициализированными данными

.bss: секция с неинициализированными глобальными переменными



Задание 04.

1. Ответьте на вопрос, соответствующий вашему варианту

Вариант 1

Вариант 1. Укажите содержимое байта по виртуальному адресу 0x01579333, если регистр CR3 содержит значение 0x13907000.

Решение:

**CR3 регистр 0x13907000(16) = 00010011100100000111000000000000(2)**

**Старшие 10 бит – индекс каталога страниц- 0000 0000 0101 – 0х005**

**Средние 10 бит – индекс таблицы страниц - 0001 0111 1001 – 0х179**

**Младшие 12 бит – смещение по странице - 0011 0011 0011 – 0х333**

**Базовый адрес страницы**

**0x13907000 + 0х005\*4 = 0x13907000 + 0x14(смещение) = 0x13907014**



**Чтобы выбрать нужное нам слово в этом дампе, смотрим на смещение (последние 12 бит), у нас это 0х014, что будет соответствовать слову 13908127**

**Физический адрес страницы = физический адрес таблицы страниц + индекс таблицы страниц\*4**

**0х13908000 + 0х179\*4 = 0x139085E0 + 0x04 = 0x139085E4**



**Берем 25024127**

**Ищем смещение**

**25024000 + 0х333 = 25024333 13**

**Ищем ближайший адрес к искомому, дальше от нашего отнимаем выбранный и**



**Ответ 75**

**Задание 05 Ответьте на следующие вопросы:**

1. **Виртуальная память** — это метод управления памятью, который использует сочетание аппаратных и программных средств для обеспечения иллюзии у каждого процесса наличия собственных адресов памяти, даже если физическая память ограничена. Виртуальная память позволяет программам работать с большими объемами данных, чем фактически доступно в системе.

**Как это работает:**

* **Виртуальные адреса**: Каждая программа работает с виртуальным адресным пространством, которое создается для неё операционной системой.
* **Соответствие адресов**: Операционная система с помощью специального модуля — диспетчера памяти — преобразует виртуальные адреса в физические адреса, соответствующие ячейкам оперативной памяти.
* **Подкачка**: Если данных для программы в физической памяти не хватает, операционная система временно перемещает неиспользуемые данные на жесткий диск или SSD в файл подкачки (paging file).

**Механизм работы:**

1. Программа запрашивает данные, используя виртуальный адрес.
2. Операционная система проверяет, находятся ли данные в оперативной памяти.
   * Если данные есть, выполняется обращение (это называется **попадание в память**, memory hit).
   * Если данных нет, происходит **промах по памяти** (page fault), и операционная система загружает нужную страницу из файла подкачки в оперативную память.
3. Для отслеживания этого процесса используется таблица страниц (page table), в которой хранится информация о соответствии виртуальных и физических адресов.

**Преимущества виртуальной памяти:**

* Увеличение доступного объема памяти для программ.
* Изоляция программ друг от друга (безопасность).
* Возможность работы с большими массивами данных.

**Недостатки:**

* **Снижение производительности**: Частое обращение к файлу подкачки на диске (явление, известное как thrashing) замедляет работу системы, так как диск значительно медленнее RAM.
* **Износ SSD**: Использование файла подкачки увеличивает количество операций записи на SSD, что может сократить его срок службы.

1. **Свопинг** (или "обмен") — это процесс перемещения данных между основной оперативной памятью и внешними носителями (например, жестким диском или SSD) для освобождения места в памяти. Когда оперативной памяти не хватает для всех данных, менее используемые данные выгружаются на диск (в файл подкачки или swap-раздел), а в освобожденной области загружаются новые данные.
2. **Страничная память** — это метод управления памятью, при котором виртуальная память разбивается на блоки фиксированного размера, называемые страницами. Каждая страница может быть размещена в произвольном месте физической памяти, а таблицы страниц используются для отображения виртуальных адресов в физические.
3. **MMU (Memory Management Unit)** — это аппаратный компонент, отвечающий за управление виртуальной памятью. MMU переводит виртуальные адреса в физические, используя таблицы страниц и другие механизмы. Она также выполняет защиту памяти, предотвращая доступ к запрещенным областям памяти.
4. **TLB (Translation Lookaside Buffer)** — это кэш-память, используемая в MMU для ускорения преобразования виртуальных адресов в физические. TLB хранит недавно используемые записи из таблиц страниц, что позволяет быстро получить физический адрес без необходимости искать его в таблице страниц.
5. **Информация в строке таблицы страниц** — каждая строка таблицы страниц содержит информацию о том, где находится физическая страница в памяти, а также информацию о защите и состоянии страницы, например, флаги доступа (чтение, запись, выполнение), бит загрузки и т.д.
6. **Применение хэш-таблиц** — хэш-таблицы используются для быстрого поиска данных с помощью функции хеширования, которая преобразует ключ в индекс массива. Это позволяет ускорить операции вставки, удаления и поиска данных по ключу.
7. **Инвертированная таблица физических страниц** — это структура данных, которая хранит информацию о том, какая виртуальная страница использует какой физический адрес. Эта таблица используется в некоторых системах для оптимизации работы с памятью, когда традиционные таблицы страниц слишком велики.
8. **Windows: области адресного пространства** — в 32-битных системах Windows адресное пространство делится на следующие области (от младших к старшим адресам):

* **Пользовательская память** (0x00000000 — 0x7FFFFFFF) — память, доступная для процессов.
* **Системная память** (0x80000000 — 0xFFFFFFFF) — зарезервирована для операционной системы. В 64-битных системах адресное пространство значительно расширено.

1. **Стандартный начальный размер области heap в Windows** — обычно это 1 МБ, но может варьироваться в зависимости от конфигурации и требований программы.
2. **Стандартный размер области памяти stack в Windows** — по умолчанию для большинства приложений это 1 МБ, но размер может быть настроен при компиляции программы.
3. **Рабочее множество** — это набор страниц памяти, которые активно используются процессом. Операционная система управляет рабочим множеством, выгружая и загружая страницы в память, чтобы поддерживать наиболее часто используемые страницы в оперативной памяти, снижая обращения к диску.
4. **Heap в Windows** — это область памяти, используемая для динамического выделения памяти во время выполнения программы. Процесс может создавать несколько куч (heap), но одна из них является стандартной и используется для обычных запросов на выделение памяти.

* **Heap процесса** — это кучи, связанные с конкретным процессом, которые управляются системой.
* **Пользовательская heap** — это куча, созданная пользователем с помощью специальных функций API, таких как HeapCreate.

1. **Linux: области адресного пространства** — в 32-битных системах Linux адресное пространство обычно делится так:

* **Пользовательское пространство** (0x00000000 — 0xBFFFFFFF) — память, доступная для процессов.
* **Ядро** (0xC0000000 — 0xFFFFFFFF) — память, зарезервированная для операционной системы. В 64-битных системах распределение значительное расширено.

1. **Linux: выделение памяти с помощью malloc и calloc** — функции malloc и calloc выделяют память в области пользовательского пространства. Выделенная память обычно находится в куче (heap), и операционная система управляет ее размером и расположением внутри этой области.