Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Российский государственный университет нефти и газа (национальный исследовательский университет) имени И. М. Губкина»

Кафедра Автоматизированных систем управления

Отчет по лабораторной работе № 3

дисциплины ***Основы организации операционных систем***

Управление памятью

Группа: АС-23-04

Студент: Ханеский Ярослав Александрович

К.т.н., доцент Фридлянд Александр Михайлович

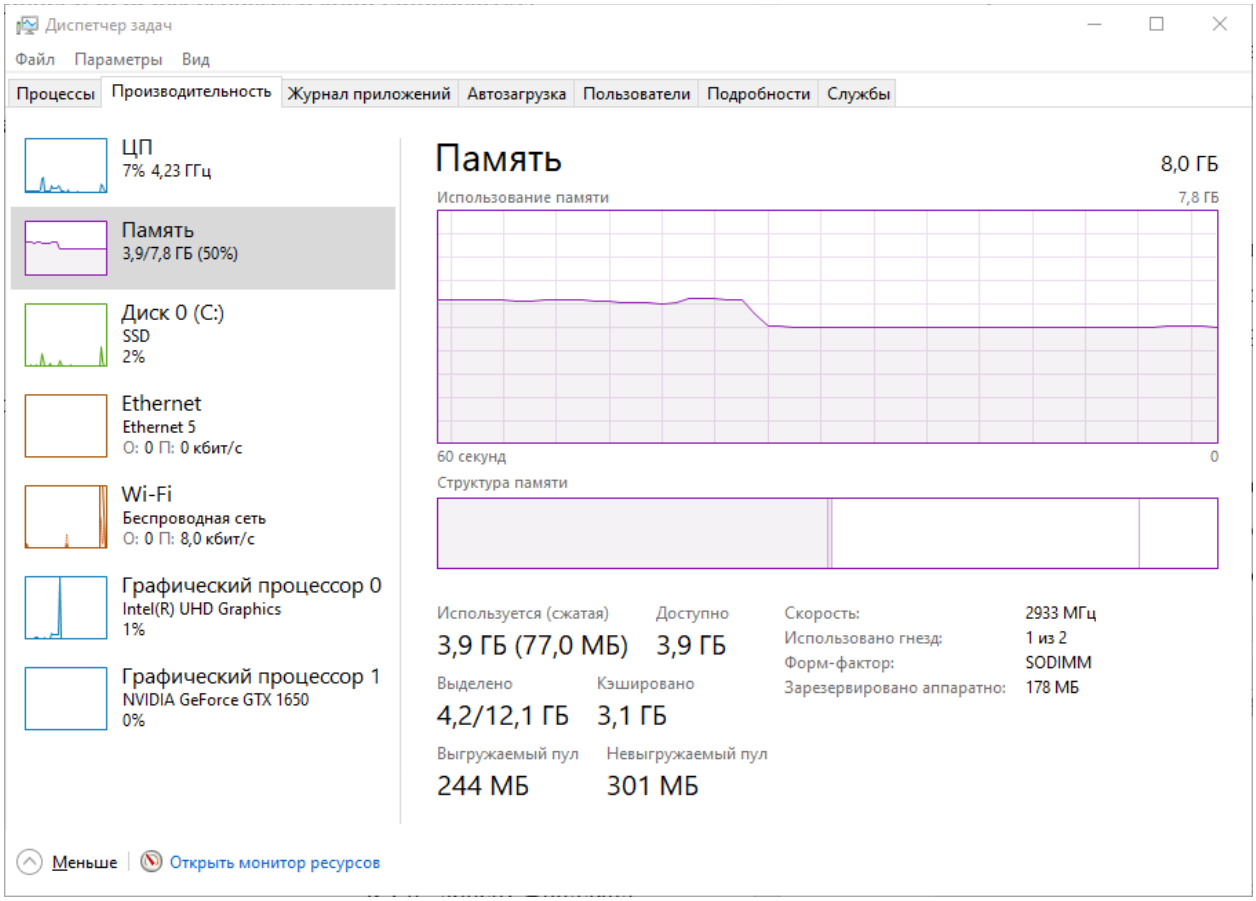
Москва

2024 г.

Цель работы: изучить принципы выделения памяти, анализ расхода памяти в процессе.

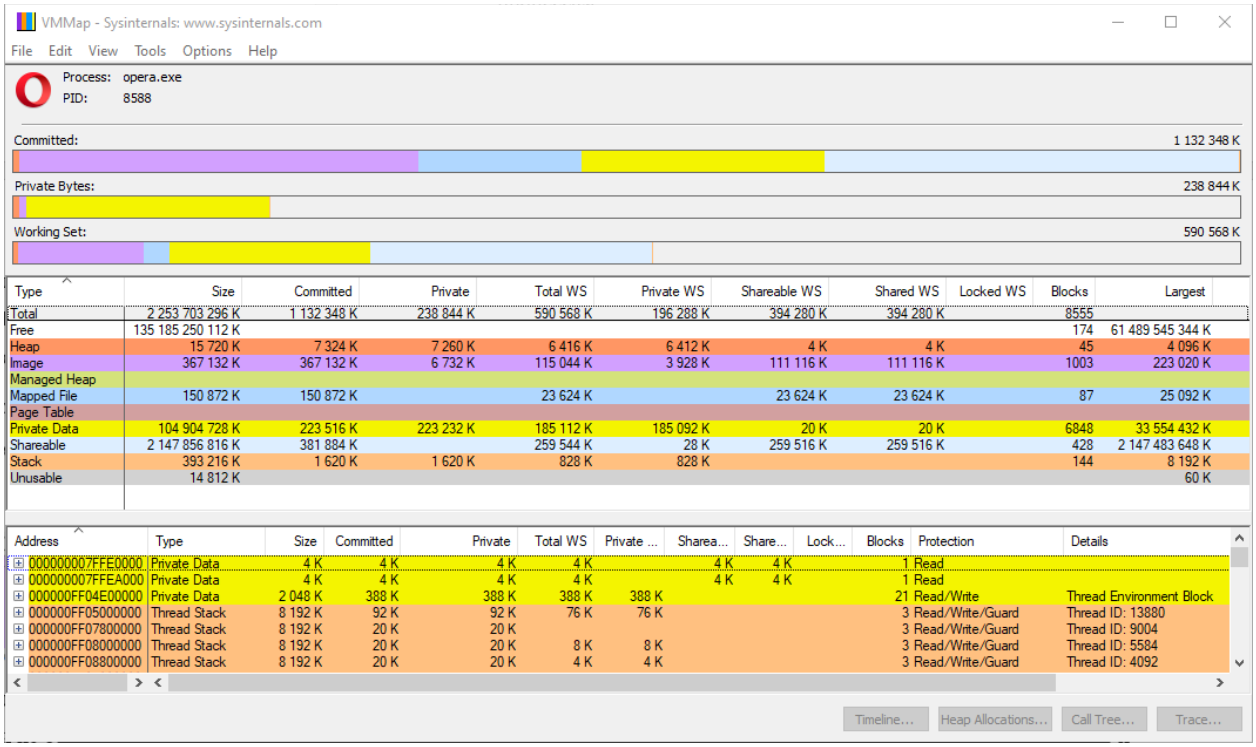
Ход работы:

1) С помощью встроенного в Windows системного ПО «Диспетчер задач» проведен анализ памяти основной машины:



В этом разделе отображены: объем ОЗУ в ГБ, объем физической памяти, используемый в данный момент системными ресурсами (в столбце «кэширование»), а также доступный объем памяти для использования процессами. Выгружаемый пул – это объем памяти ядра и драйвера устройства, который может перетекать из физической памяти в файл подкачки, а невыгружаемый – должен оставаться в физической памяти.

2) С помощью программы VMMap проанализируем выделение памяти процессам:



На рисунке представлена общая информация по типам выделенной памяти для процесса. К типам памяти относятся Shareable, Private Data, Image, Heap Mapped File, Stack, Page Table, также отображается объем свободной памяти.

В строке Total видно, что всего процесс зарезервировал для себя 2,1 Гб памяти, при этом закомитил из них 1106 Мб, а Working Set у него 577 Мб.

Рассмотрим каждый тип памяти по отдельности:

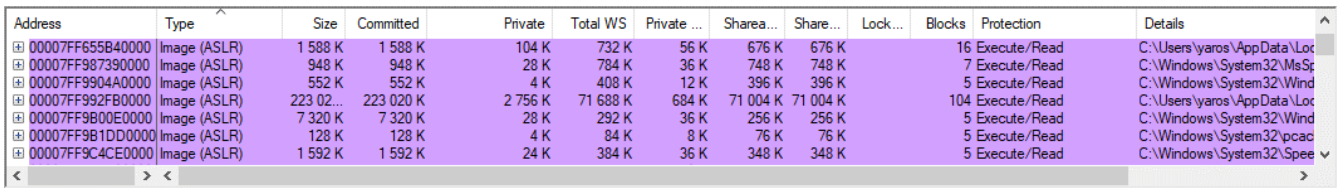


Image – память, отданная под dll и exe-файлы. Каждая dll должна быть замаплена на память приложения, так что хоть она и не будет грузиться с диска или копироваться в памяти, но она займет некоторую часть виртуальной памяти приложения. Кроме того, многие dll выделяют внутри себя память, так что они еще и расходуют Working Set приложения.

Address	Type	Size	Committed	Private	Total WS	Private ...	Sharea...	Share...	Lock...	Blocks	Protection	Details
000002B2826E0000	Mapped File	10 256 K	10 256 K		1 348 K		1 348 K	1 348 K		1	Read	C:\Users\yaros\AppData\Loc
000002B2834F0000	Mapped File	3 296 K	3 296 K		100 K		100 K	100 K		1	Read	C:\Windows\Globalization\Sc
000002B283830000	Mapped File	1 044 K	1 044 K		204 K		204 K	204 K		1	Read	C:\Users\yaros\AppData\Loc
000002B283940000	Mapped File	1 100 K	1 100 K		28 K		28 K	28 K		1	Read	C:\Users\yaros\AppData\Loc
000002B283A60000	Mapped File	1 228 K	1 228 K		28 K		28 K	28 K		1	Read	C:\Users\yaros\AppData\Loc
000002B283BA0000	Mapped File	1 612 K	1 612 K		32 K		32 K	32 K		1	Read	C:\Users\yaros\AppData\Loc
000002B283D40000	Mapped File	1 356 K	1 356 K		32 K		32 K	32 K		1	Read	C:\Users\yaros\AppData\Loc

Mapped Files относится к памяти, которая связана с отображенными в память файлами (memory-mapped files). Этот механизм позволяет приложениям работать с содержимым файла так, будто оно находится в оперативной памяти, что упрощает и ускоряет доступ к данным.

Address	Type	Size	Committed	Private	Total WS	Private ...	Sharea...	Share...	Lock...	Blocks	Protection	Details
0000004A80000000	Thread Stack	8 192 K	40 K	40 K	28 K	28 K				3	Read/Write/Guard	Thread ID: 11488
0000004A80800000	Thread Stack	8 192 K	52 K	52 K	40 K	40 K				3	Read/Write/Guard	Thread ID: 8792
0000004A81000000	Thread Stack	8 192 K	24 K	24 K	12 K	12 K				3	Read/Write/Guard	Thread ID: 2248
0000004A81800000	Thread Stack	8 192 K	52 K	52 K	40 K	40 K				3	Read/Write/Guard	Thread ID: 3236
0000004A82000000	Thread Stack	8 192 K	28 K	28 K	16 K	16 K				3	Read/Write/Guard	Thread ID: 11616
0000004A83000000	Thread Stack	8 192 K	32 K	32 K	20 K	20 K				3	Read/Write/Guard	Thread ID: 14972
0000004A83800000	Thread Stack	8 192 K	20 K	20 K	8 K	8 K				3	Read/Write/Guard	Thread ID: 8412

Stack – часть памяти, выделенная для каждого потока процесса, связанных с выполнением функций.

Address	Type	Size	Committed	Private	Total WS	Private ...	Sharea...	Share...	Lock...	Blocks	Protection	Details
000001B587CC0000	Heap (Private Data)	392 K	64 K	64 K	64 K	64 K				2	Read/Write	Heap ID: 1 [LOW FRAGMEN
000001B587D40000	Heap (Private Data)	1 024 K	1 020 K	1 020 K	1 000 K	1 000 K				2	Read/Write	Heap ID: 1 [LOW FRAGMEN
000001B587F10000	Heap (Private Data)	392 K	8 K	8 K	8 K	8 K				2	Read/Write	Heap ID: 3 [LOW FRAGMEN
000001B587F90000	Heap (Private Data)	64 K	8 K	8 K	8 K	8 K				2	Read	Heap ID: 7 [COMPATABILIT
000001B587FC0000	Heap (Private Data)	64 K	8 K	8 K	8 K	8 K				2	Read/Write	Heap ID: 5 [COMPATABILIT
000001B5880F0000	Heap (Private Data)	64 K	60 K	60 K	60 K	60 K				2	Read/Write	Heap ID: 3 [LOW FRAGMEN
000001B58A830000	Heap (Private Data)	64 K	60 K	60 K	52 K	52 K				2	Read/Write	Heap ID: 4 [LOW FRAGMEN

Heap – динамическая область памяти, из которой приложения могут выделять и освобождать блоки во время выполнения. Эта память используется для хранения данных, которые не могут быть заранее предсказаны или имеют переменный размер.

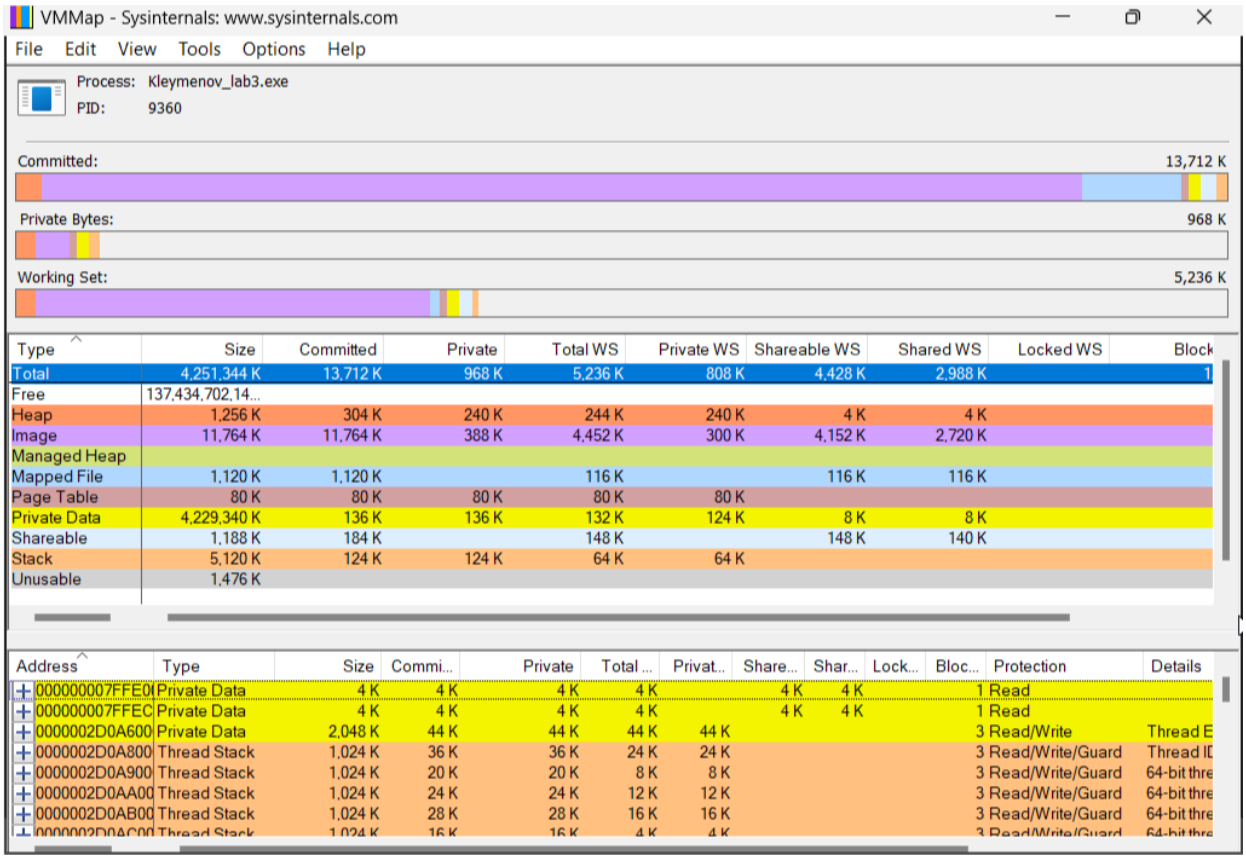
Address	Type	Size	Committed	Private	Total WS	Private ...	Sharea...	Share...	Lock...	Blocks	Protection	Details
00000007FFE0000	Private Data	4 K	4 K	4 K	4 K		4 K	4 K		1	Read	
00000007FFEA000	Private Data	4 K	4 K	4 K	4 K		4 K	4 K		1	Read	
0000004AFA200000	Private Data	2 048 K	364 K	364 K	360 K	360 K				13	Read/Write	Thread Environment Block
000001B587C90000	Private Data	8 K	8 K	8 K	8 K	8 K				1	Read/Write	
000001B5880E0000	Private Data	4 K	4 K	4 K	4 K	4 K				1	Read/Write	
000001B58A2B0000	Private Data	4 096 K	4 096 K	4 096 K	576 K	576 K				1	Read/Write	
000001B58A7D0000	Private Data	4 K	4 K	4 K	4 K	4 K				1	Read/Write	

Private – это уникальная область памяти, доступная только текущему процессу. Она используется для хранения данных, которые являются специфичными для приложения и не предназначены для совместного использования.

Address	Type	Size	Committed	Private	Total WS	Private ...	Sharea...	Share...	Lock...	Blocks	Protection	Details
000001B587C40000	Shareable	4 K	4 K		4 K		4 K	4 K		1	Read	
000001B587C50000	Shareable	116 K	116 K		112 K		112 K	112 K		1	Read	
000001B587C70000	Shareable	16 K	16 K		12 K		12 K	12 K		1	Read	
000001B587C80000	Shareable	12 K	12 K		12 K		12 K	12 K		1	Read	
000001B587CA0000	Shareable	4 K	4 K		4 K		4 K	4 K		1	Read	
000001B587CB0000	Shareable	4 K	4 K		4 K		4 K	4 K		1	Read	
000001B587D30000	Shareable	32 K	16 K		12 K		12 K	12 K		2	Read	

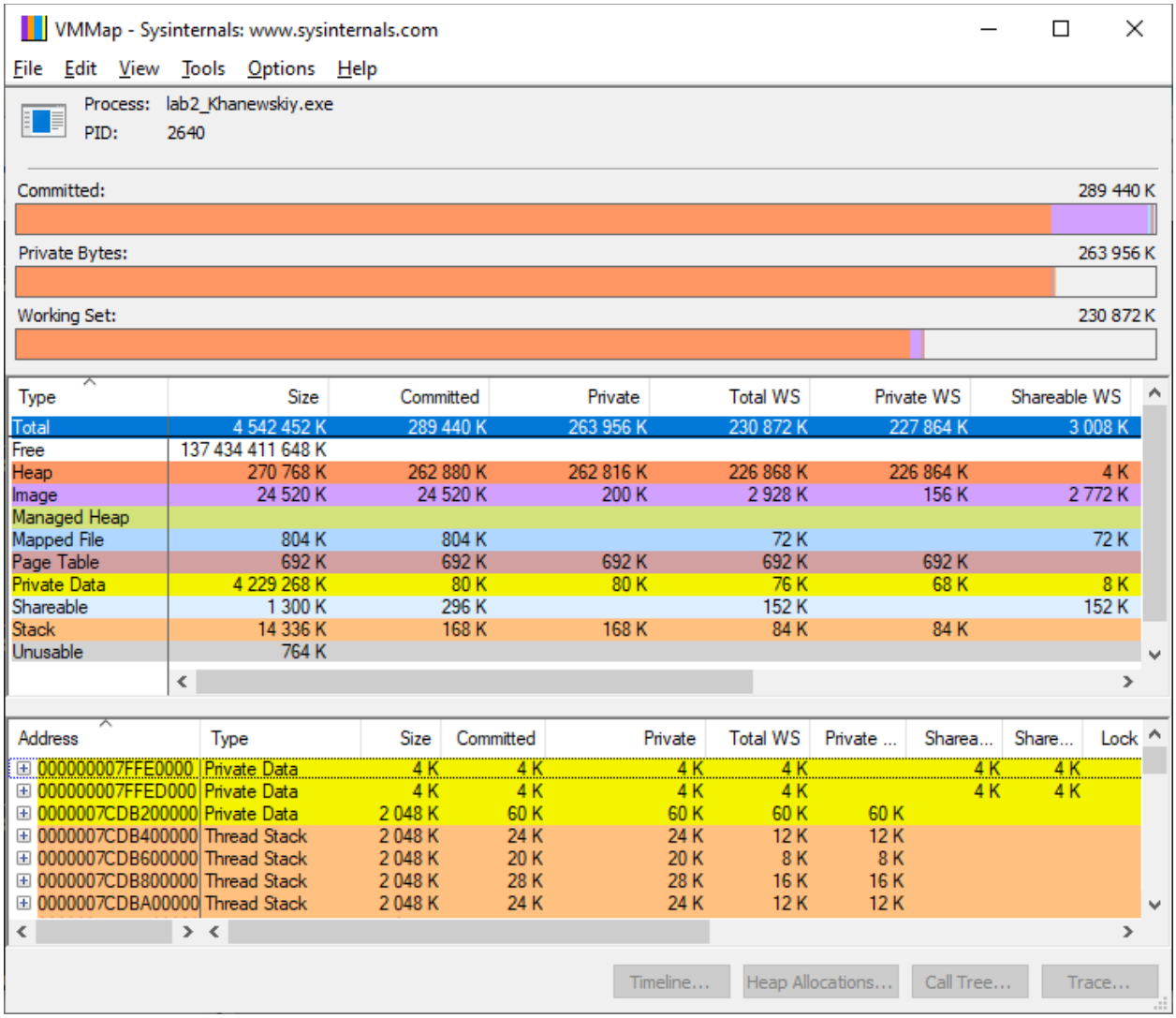
Shareable – память, которая может быть использована несколькими процессами одновременно. Это данные, которые находятся в общем доступе и позволяют нескольким процессам читать или, реже, изменять их содержимое.

3) Запустим программу из лабораторной работы №2 в VMMap и проведем полный анализ всех ресурсов:



Type	Size	Committed	Private	Total WS	Private WS	Shareable WS	Shared WS	Locked WS	Block
Total	4,251,344 K	13,712 K	968 K	5,236 K	808 K	4,428 K	2,988 K		1
Free	137,434,702,14...								
Heap	1,256 K	304 K	240 K	244 K	240 K	4 K	4 K		
Image	11,764 K	11,764 K	388 K	4,452 K	300 K	4,152 K	2,720 K		
Managed Heap									
Mapped File	1,120 K	1,120 K		116 K		116 K	116 K		
Page Table	80 K	80 K	80 K	80 K	80 K				
Private Data	4,229,340 K	136 K	136 K	132 K	124 K	8 K	8 K		
Shareable	1,188 K	184 K		148 K		148 K	140 K		
Stack	5,120 K	124 K	124 K	64 K	64 K				
Unusable	1,476 K								

Address	Type	Size	Commi...	Private	Total ...	Privat...	Share...	Shar...	Lock...	Bloc...	Protection	Details
000000007FFE0(	Private Data	4 K	4 K	4 K	4 K		4 K	4 K		1	Read	
000000007FFEC	Private Data	4 K	4 K	4 K	4 K		4 K	4 K		1	Read	
0000002D0A600	Private Data	2,048 K	44 K	44 K	44 K	44 K				3	Read/Write	Thread E
0000002D0A800	Thread Stack	1,024 K	36 K	36 K	24 K	24 K				3	Read/Write/Guard	Thread ID
0000002D0A900	Thread Stack	1,024 K	20 K	20 K	8 K	8 K				3	Read/Write/Guard	64-bit thre
0000002D0AA00	Thread Stack	1,024 K	24 K	24 K	12 K	12 K				3	Read/Write/Guard	64-bit thre
0000002D0AB00	Thread Stack	1,024 K	28 K	28 K	16 K	16 K				3	Read/Write/Guard	64-bit thre



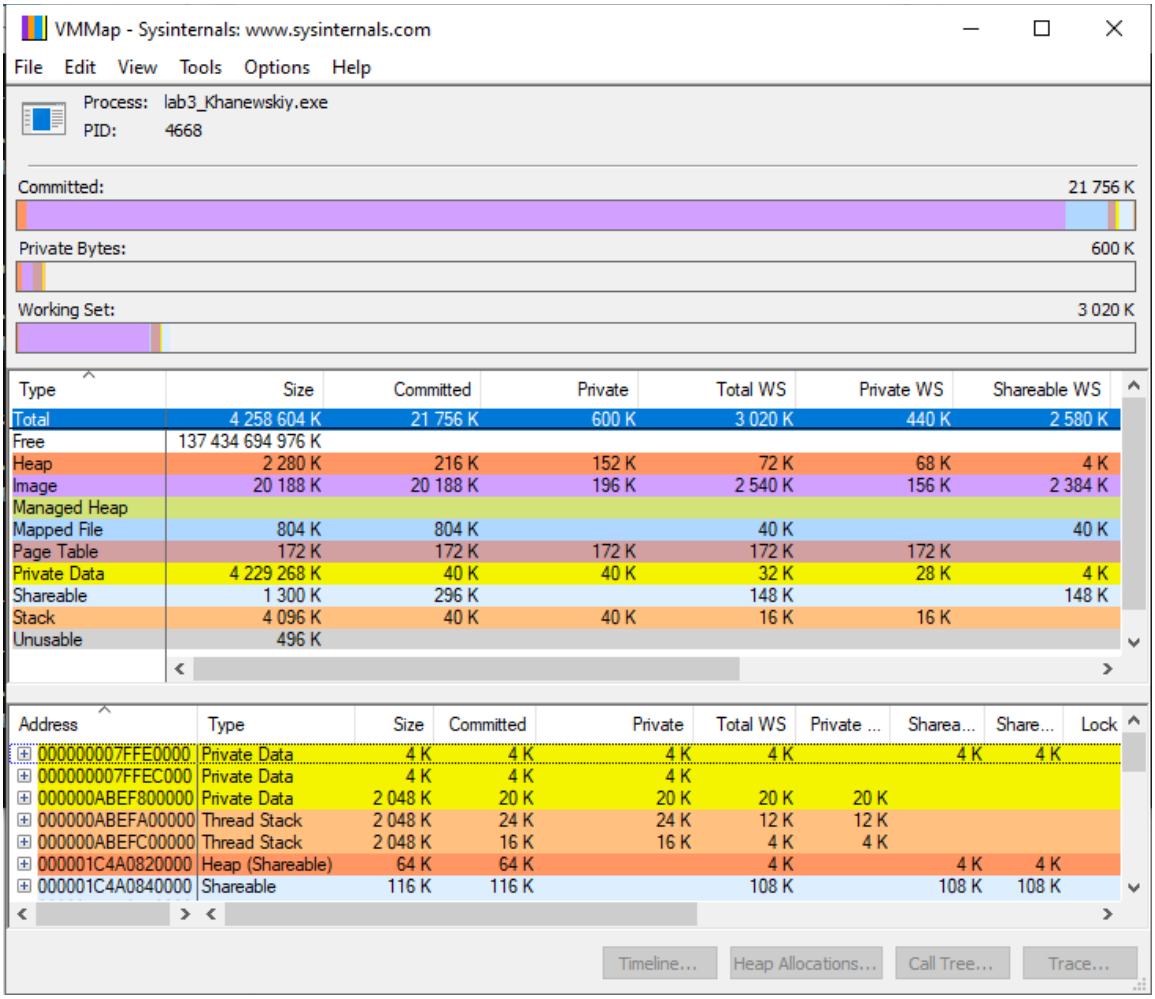
Больше всего памяти занимает тип Heap, так как во время выполнения программы в экземпляры класса stringstream помещаются полное содержимое текстовых файлов. После него идет тип Image, так как он содержит исполняемый код и статические данные программы. Далее память выделяется для каждого потока процесса, связанного с выполнением функций(Stack). Private Data уходит на стеки, кучи и другие приватные структуры данных. Оставшаяся память распределяется между Shareable, Mapped File и Page Table.

4) Реализуем программу для изучения фрагментации виртуальной памяти:

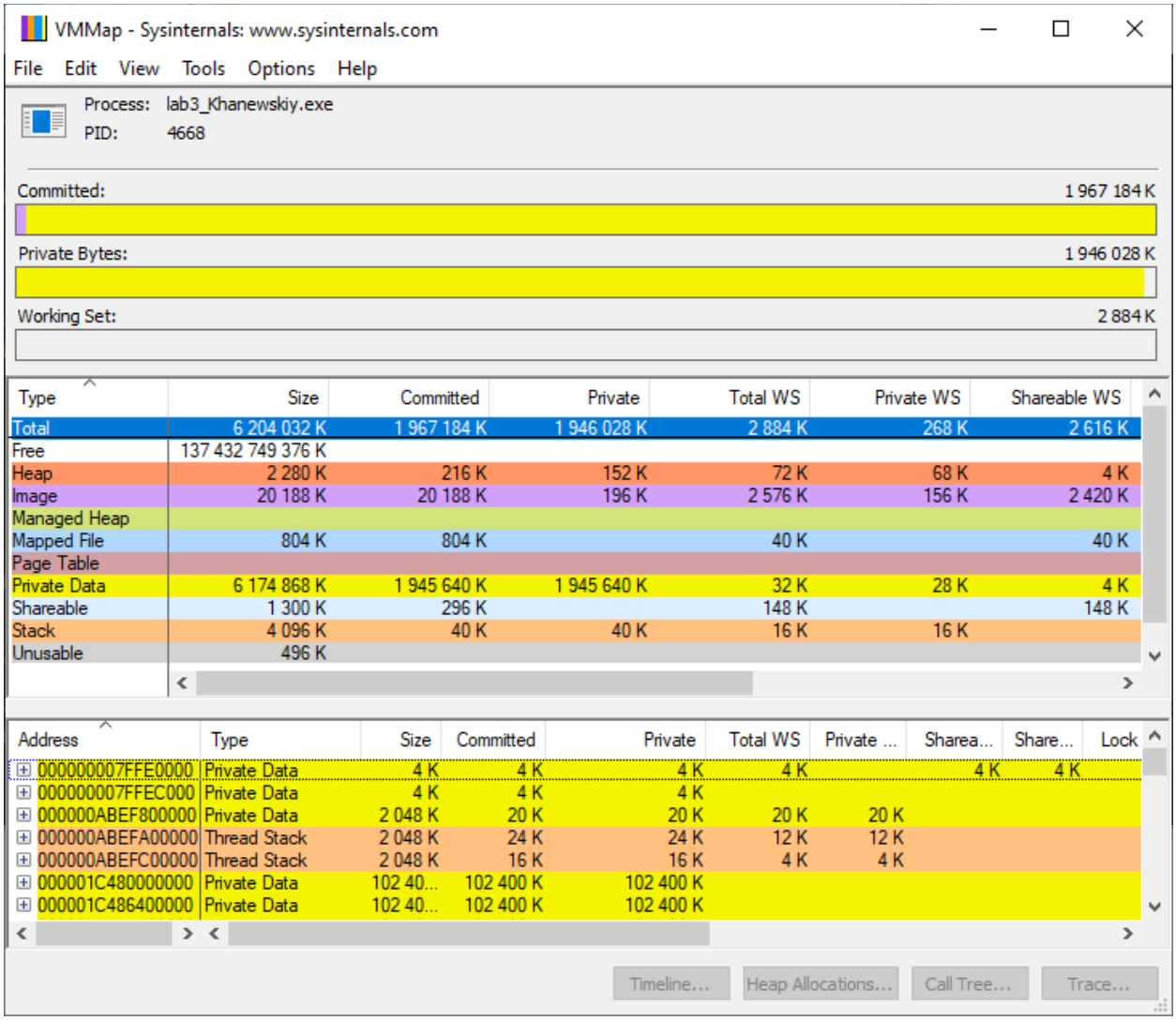
```cpp
#include <windows.h>
#include <iostream>
#include <vector>
#include <iomanip>
#include <unistd.h>

void printMemoryStatus() {
    MEMORYSTATUSEX memStatus = {};
    memStatus.dwLength = sizeof(MEMORYSTATUSEX);
    if (GlobalMemoryStatusEx(&memStatus)) {
        std::cout << "Memory Status:\n";
        std::cout << "  Total Virtual Memory: " << memStatus.ullTotalPageFile / (1024 * 1024) << " MB\n";
        std::cout << "  Available Virtual Memory: " << memStatus.ullAvailPageFile / (1024 * 1024) << " MB\n";
        std::cout << "  Memory Load: " << memStatus.dwMemoryLoad << " %\n\n";
    } else {
        std::cerr << "Failed to get memory status.\n";
    }

    sleep(15);
}

int main() {
    std::vector<void*> memoryBlocks;
    const size_t blockSize = 100 * 1024 * 1024; // 100 MB

    //Step 1
    printMemoryStatus();

    // Step 2: Allocate all available virtual memory in 100 MB blocks
    while (true) {
        void* block = VirtualAlloc(nullptr, blockSize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
        if (block) {
            memoryBlocks.push_back(block);
        } else {
            break; // Stop when memory allocation fails
        }
    }

    std::cout << "Allocated " << memoryBlocks.size() << " blocks of 100 MB.\n";

    //Step 3
    printMemoryStatus();

    // Step 4: Free every second block
    for (size_t i = 0; i < memoryBlocks.size(); i += 2) {
        if (memoryBlocks[i]) {
            VirtualFree(memoryBlocks[i], 0, MEM_RELEASE);
            memoryBlocks[i] = nullptr;
        }
    }

    std::cout << "Freed every second block.\n";

    //Step 5
    printMemoryStatus();

    // Step 6: Try to allocate a block of available memory
    MEMORYSTATUSEX memStatus = {};
    memStatus.dwLength = sizeof(MEMORYSTATUSEX);
    GlobalMemoryStatusEx(&memStatus);
    size_t availableMemory = memStatus.ullAvailPageFile;
    void* largeBlock = VirtualAlloc(nullptr, availableMemory, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

    if (largeBlock) {
        std::cout << "Successfully allocated a large block of size: "
                  << availableMemory / (1024 * 1024) << " MB.\n";
        VirtualFree(largeBlock, 0, MEM_RELEASE);
    } else {
        std::cerr << "Failed to allocate a large block of memory.\n";
```

```
    }

    // Step 7: Free remaining blocks
    for (void* block : memoryBlocks) {
        if (block) {
            VirtualFree(block, 0, MEM_RELEASE);
        }
    }

    std::cout << "Freed all remaining blocks.\n";

    // Step 8
    printMemoryStatus();

    // Step 9: Reserve a block of memory of size equal to available memory
    void* reservedBlock = VirtualAlloc(nullptr, availableMemory, MEM_RESERVE, PAGE_NOACCESS);
    if (reservedBlock) {
        std::cout << "Reserved a block of size: " << availableMemory / (1024 * 1024) << " MB.\n";
        VirtualFree(reservedBlock, 0, MEM_RELEASE);
    } else {
        std::cerr << "Failed to reserve a block of memory.\n";
    }

    printMemoryStatus();
    return 0;
}
```

Выделение памяти на пункте 1:

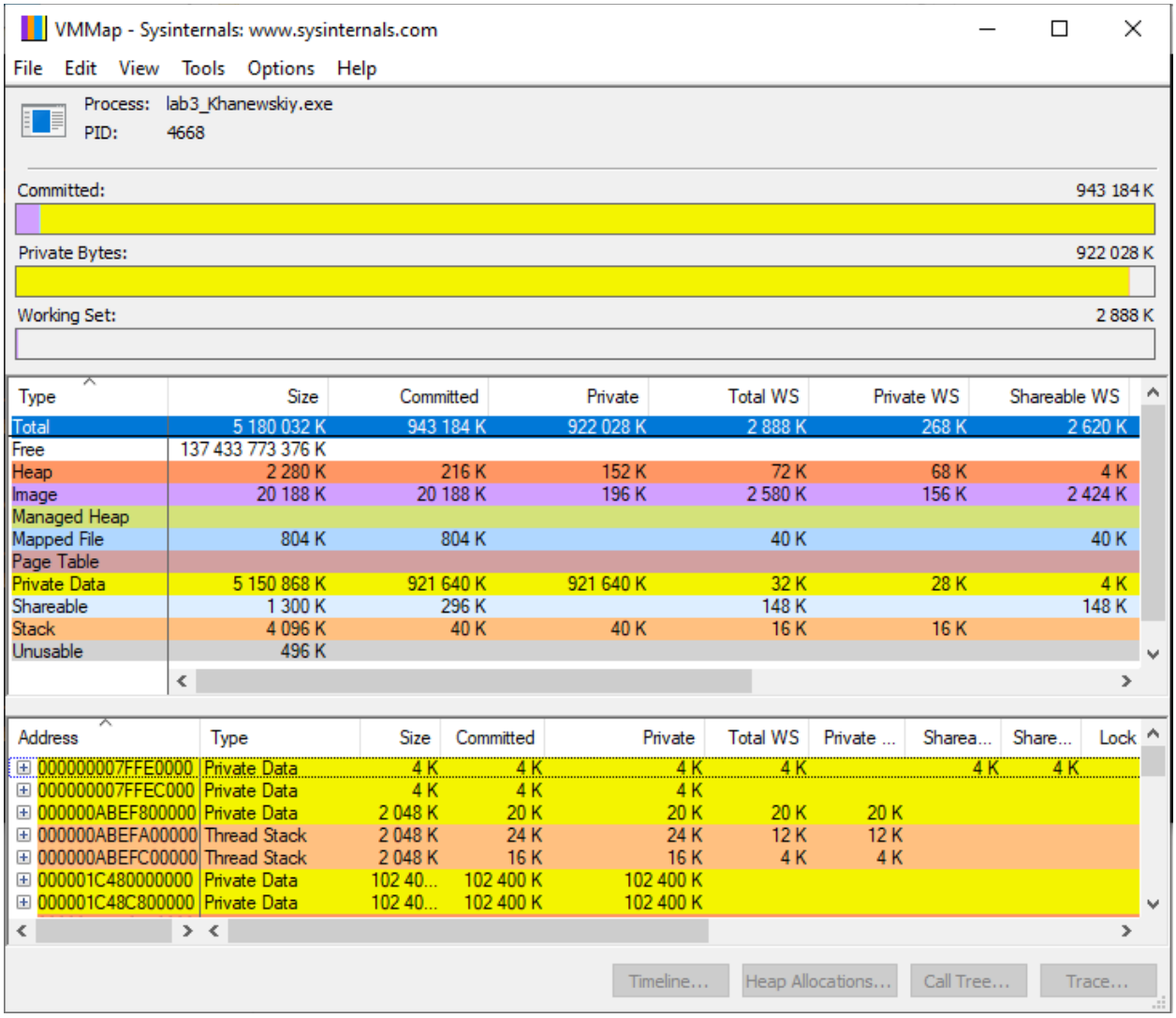


Выделение памяти на пункте 3:



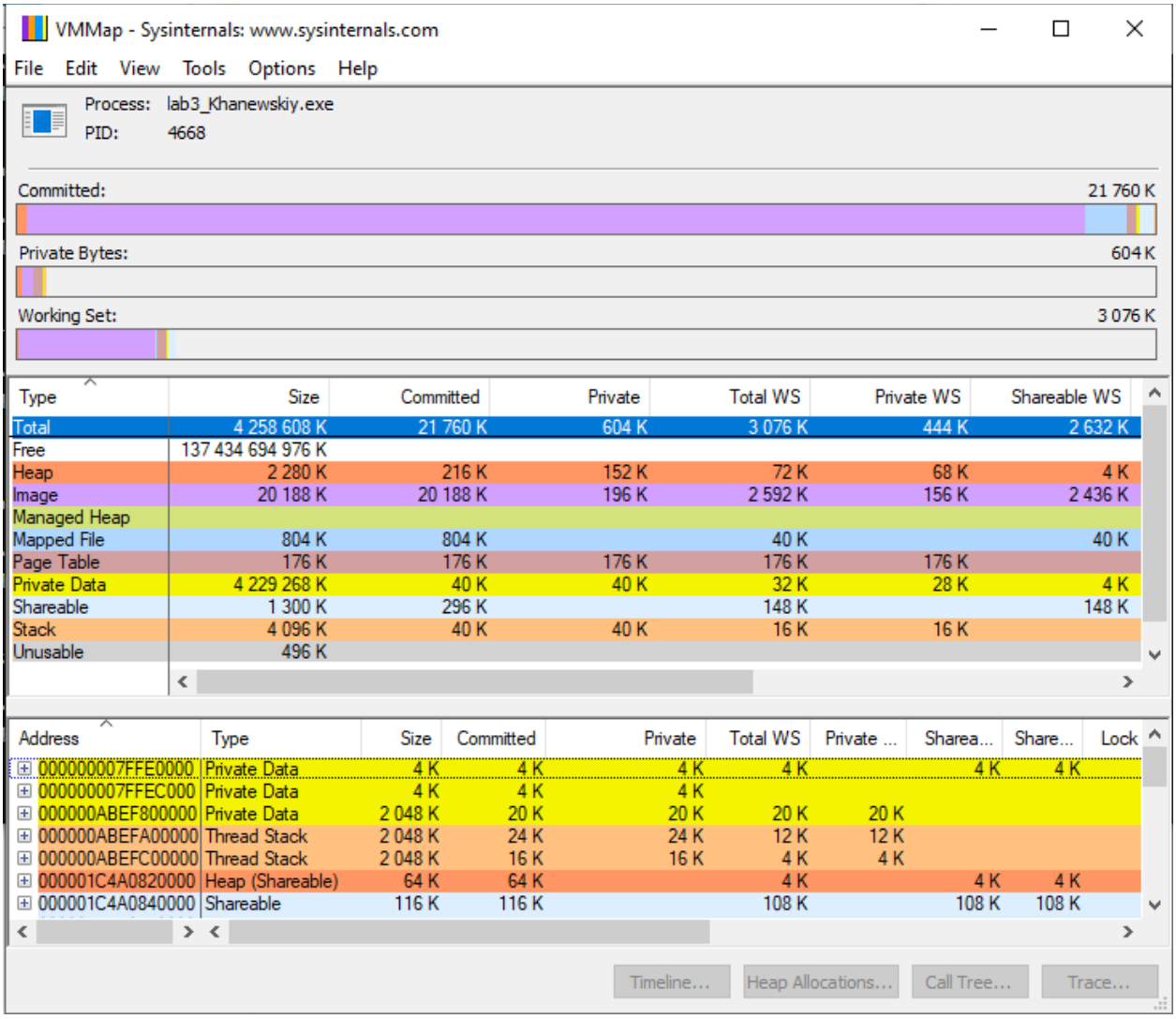
После использования всей доступной ВП, выделяемой функцией VirtualAlloc() областей размером по 100 МБ, видно, что увеличился как общий размер Private Data, так и размер закоммиченной области.

Выделение памяти на пункте 5:



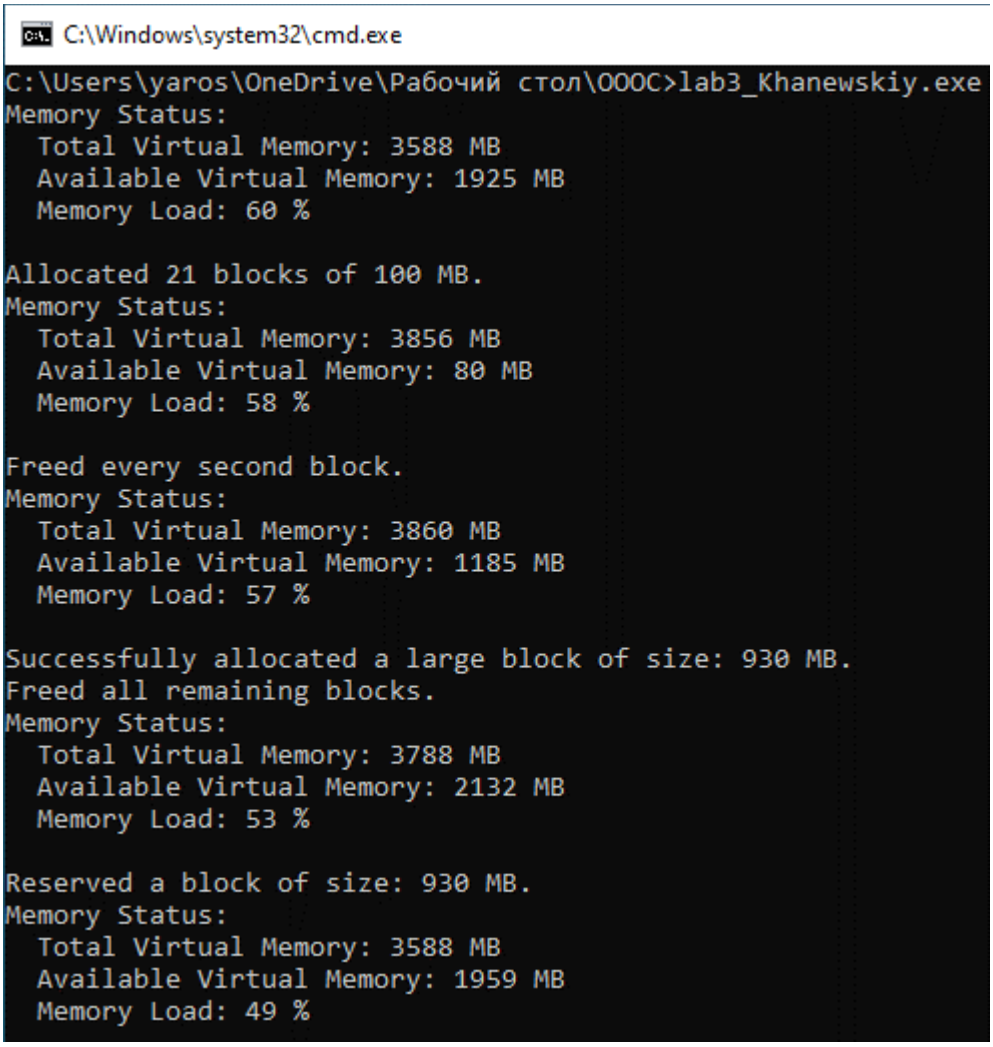
Освобождение каждого второго из выделенных в пункте 2 блоков при помощи функции VirtualFree () привело к уменьшению закоммиченной области Private Data чуть больше, чем в 2 раза, а также к уменьшению на такой же объём общего размера Private Data. Также в 2 раза уменьшился размер Stack, так как в векторе memoryBlocks, хранящем указатели на функции, был удалён каждый второй элемент.

Выделение памяти на пункте 8:



Освобождение всех блоков привело к полному очищению закоммиченной области Private Data и к уменьшению на такой же объём общего размера Private Data.

Вывод программы:


C:\Windows\system32\cmd.exe
C:\Users\yaros\OneDrive\Рабочий стол\ОООС>lab3_Khanewskiy.exe
Memory Status:
Total Virtual Memory: 3588 MB
Available Virtual Memory: 1925 MB
Memory Load: 60 %
Allocated 21 blocks of 100 MB.
Memory Status:
Total Virtual Memory: 3856 MB
Available Virtual Memory: 80 MB
Memory Load: 58 %
Freed every second block.
Memory Status:
Total Virtual Memory: 3860 MB
Available Virtual Memory: 1185 MB
Memory Load: 57 %
Successfully allocated a large block of size: 930 MB.
Freed all remaining blocks.
Memory Status:
Total Virtual Memory: 3788 MB
Available Virtual Memory: 2132 MB
Memory Load: 53 %
Reserved a block of size: 930 MB.
Memory Status:
Total Virtual Memory: 3588 MB
Available Virtual Memory: 1959 MB
Memory Load: 49 %