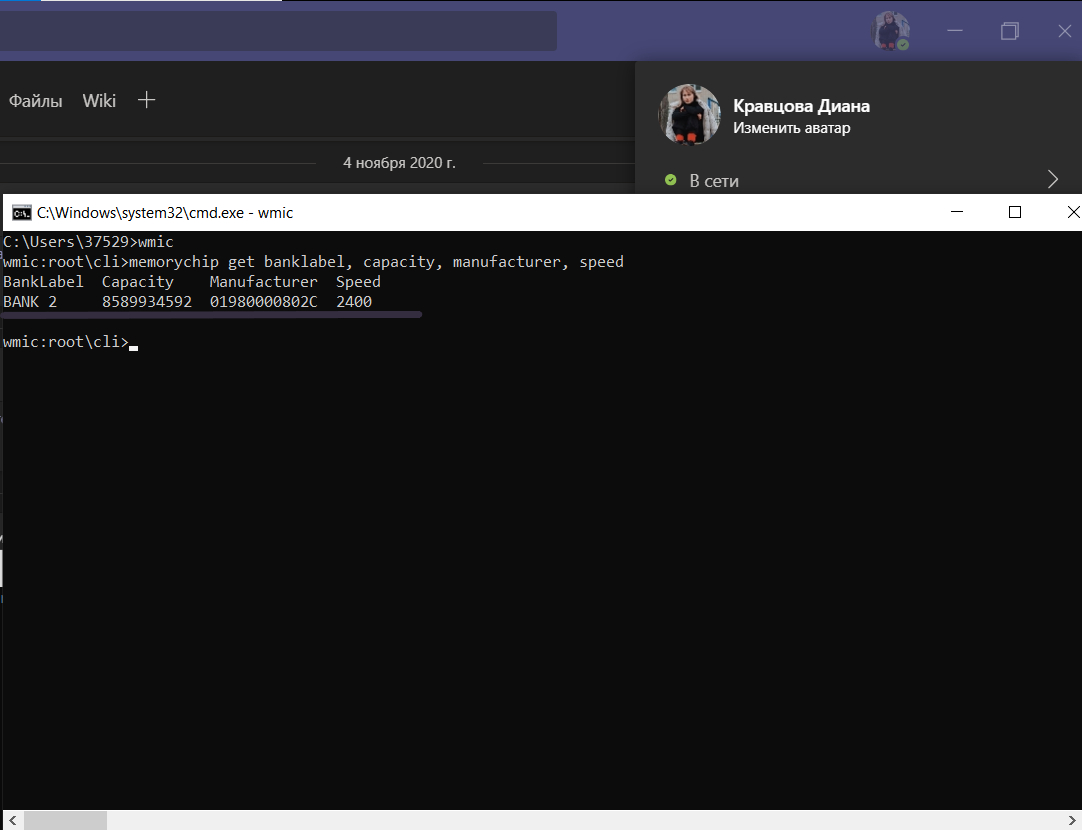
**ЗАДАНИЕ 1**

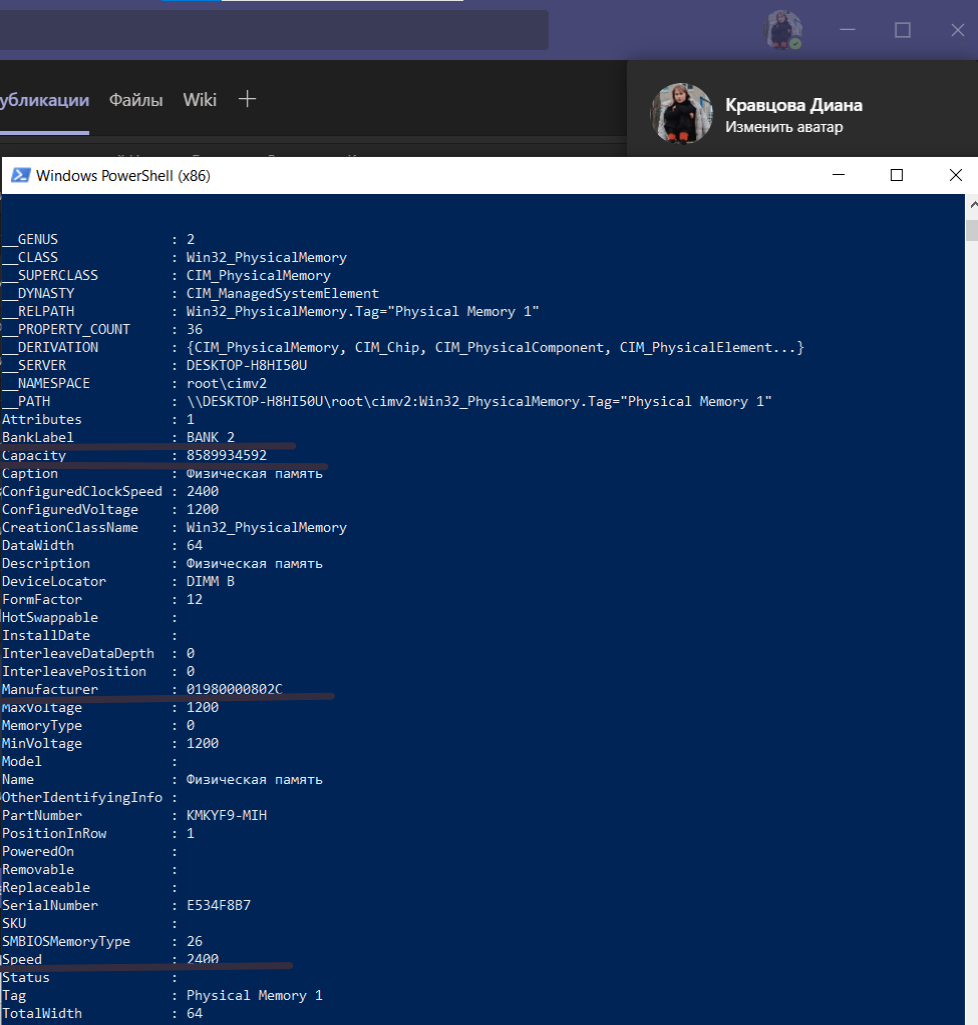


**BankLabel – разъем, в который встроен модуль памяти;**

**Capacity – объем, установленного модуля памяти;**

**Manufacturer – производитель модуля памяти;**

**Speed – скоро в МГц для каждой планки.**

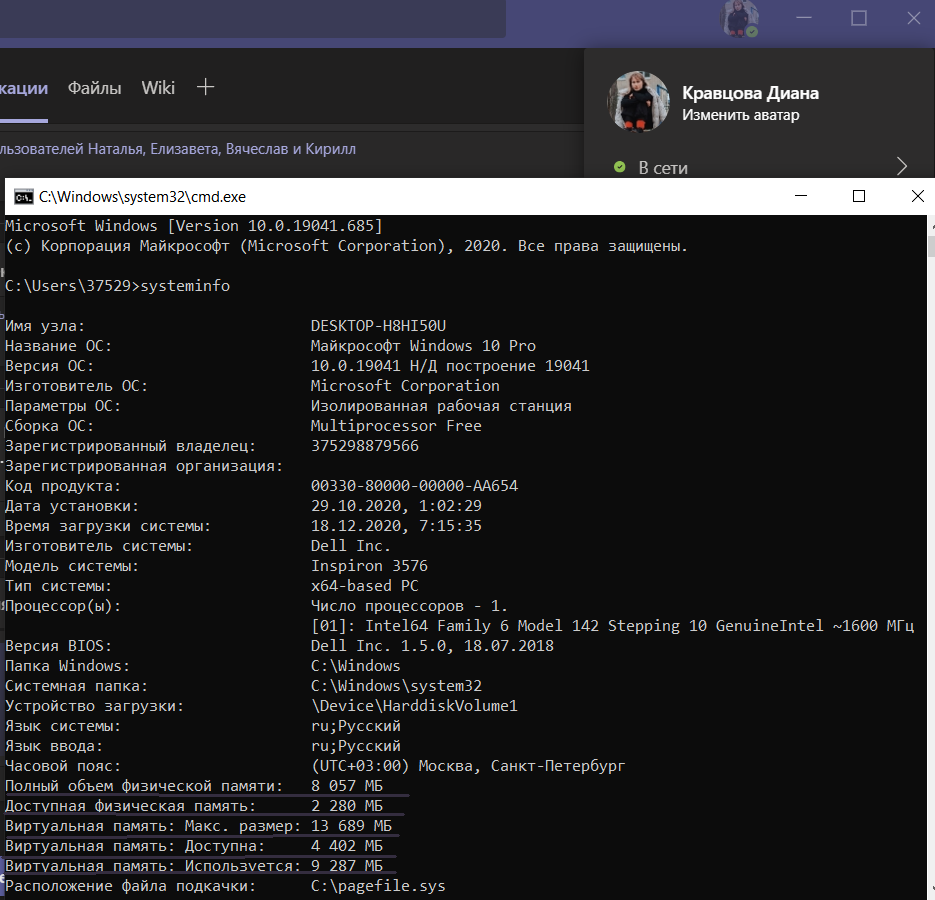


**BankLabel – разъем, в который встроен модуль памяти;**

**Capacity – объем, установленного модуля памяти;**

**Manufacturer – производитель модуля памяти;**

**Speed – скоро в МГц для каждой планки.**



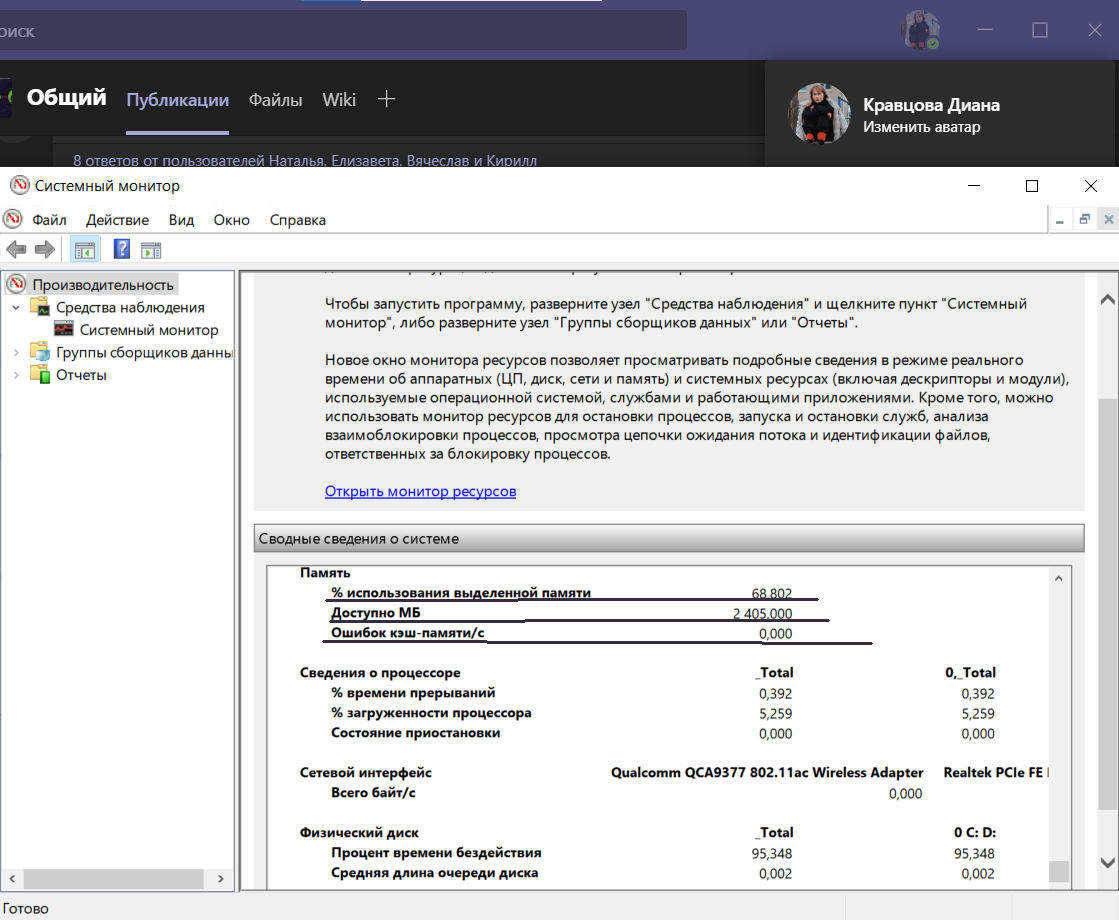
**Полный объем физической памяти в МБ;**

**Доступный объем физической памяти в МБ;**

**Максимальный объем виртуальной памяти в МБ;**

**Доступный объем виртуальной памяти в МБ;**

**Используемый объем виртуальной памяти в МБ.**



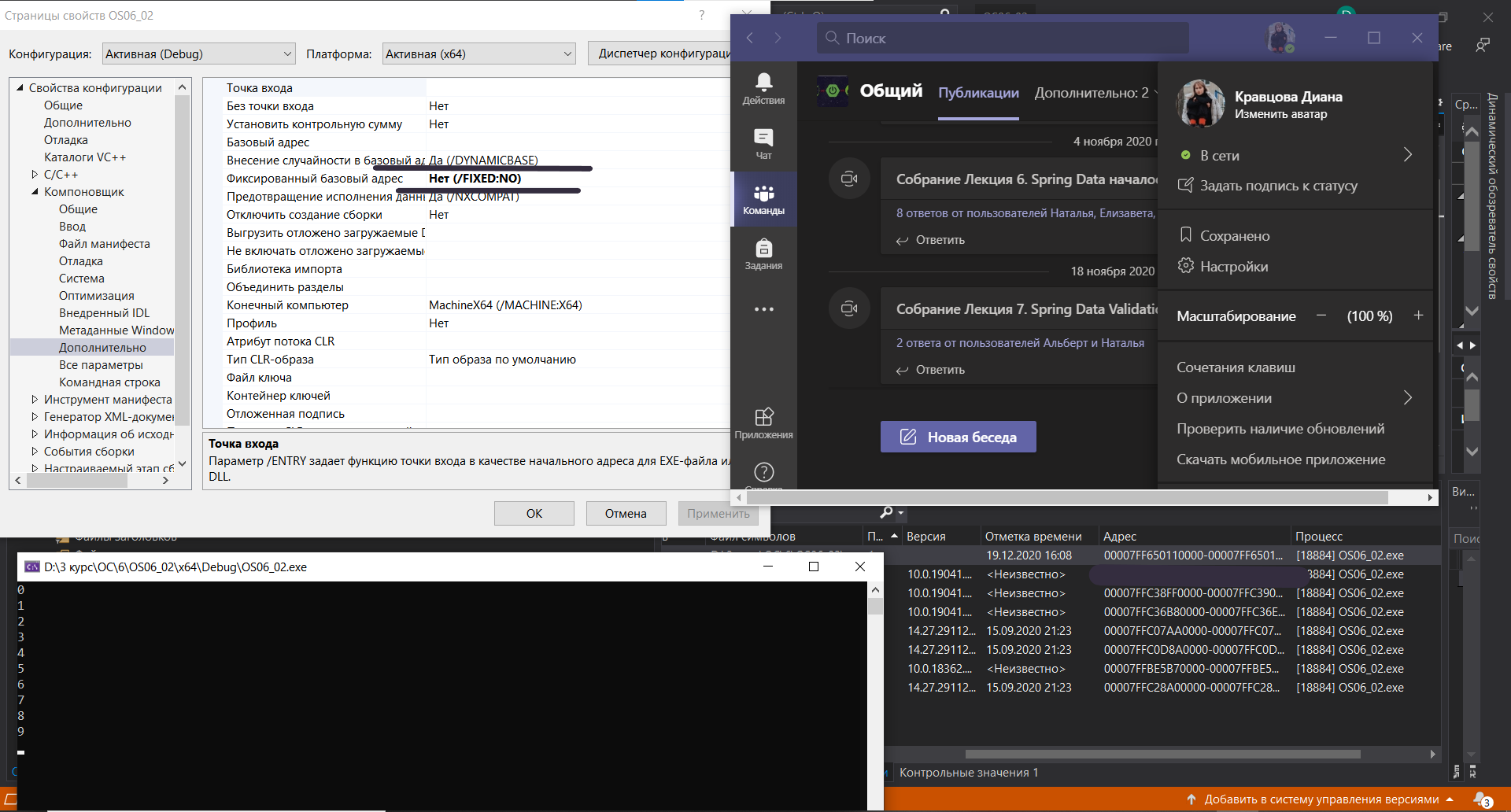
**Процент использования выделенной памяти;**

**Доступный объем физической памяти;**

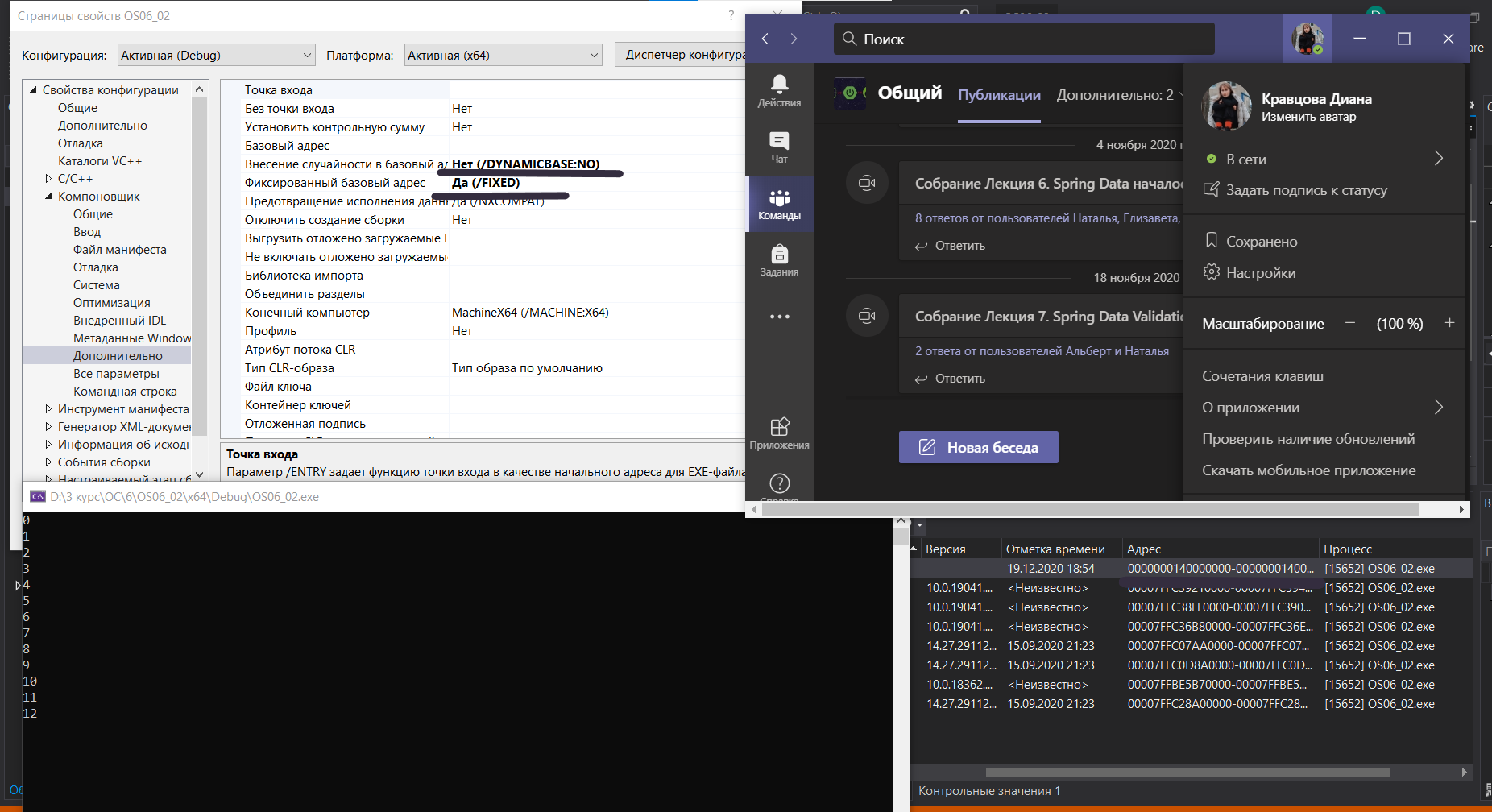
**Количество ошибок кэширования памяти в секунду.**

**ЗАДАНИЕ 2**

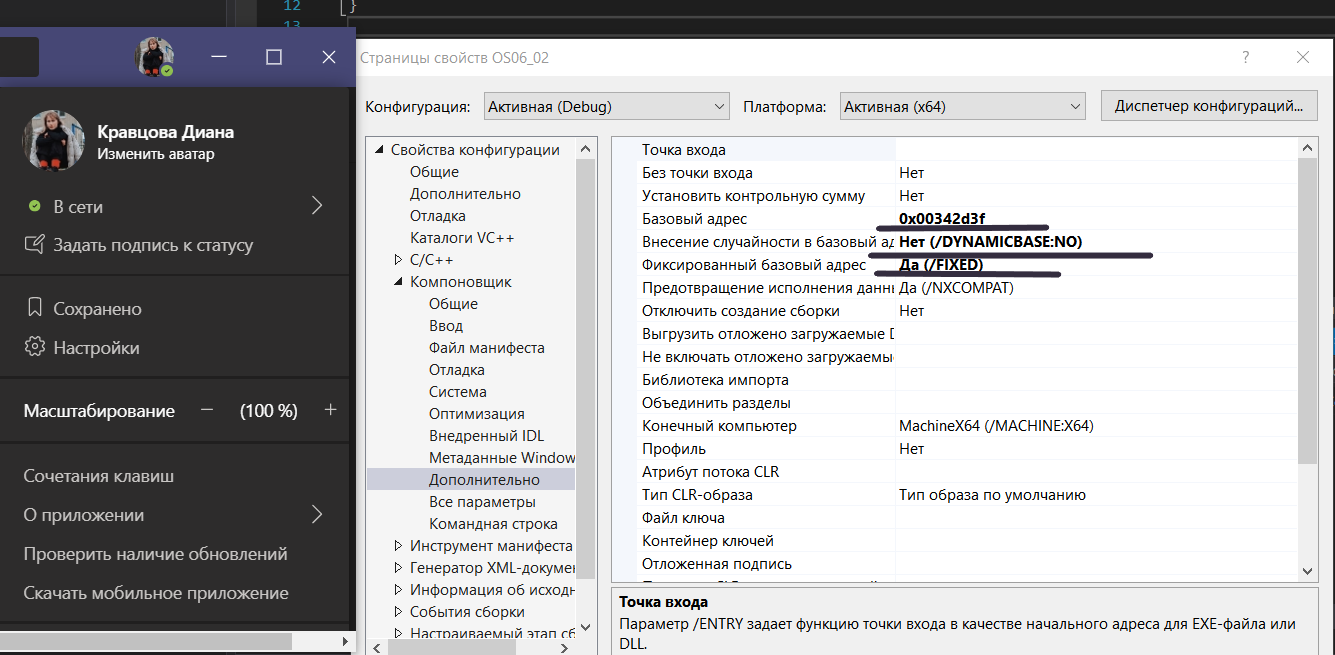
**Скриншот с выделением адрес модуля приложения без установки базового.**

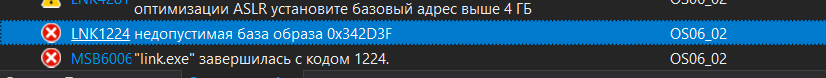


**Скриншот с выделением адрес модуля приложения с установкой стандартного базового адреса.**



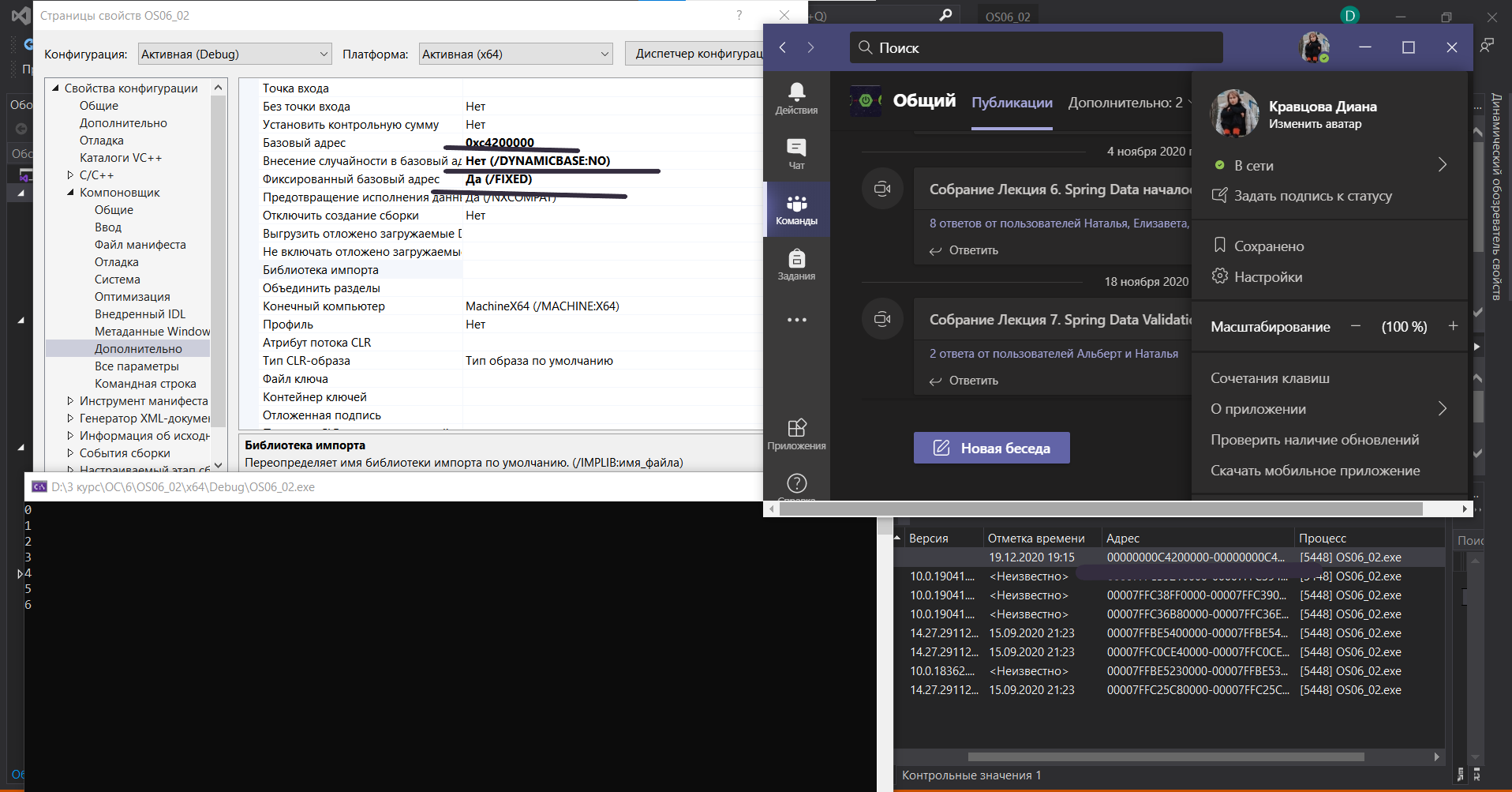
**Скриншот с выделением адрес модуля приложения с установки базового в виде 0x00{первые буквы ФИО в hex}. (0x00342d3f)**



****

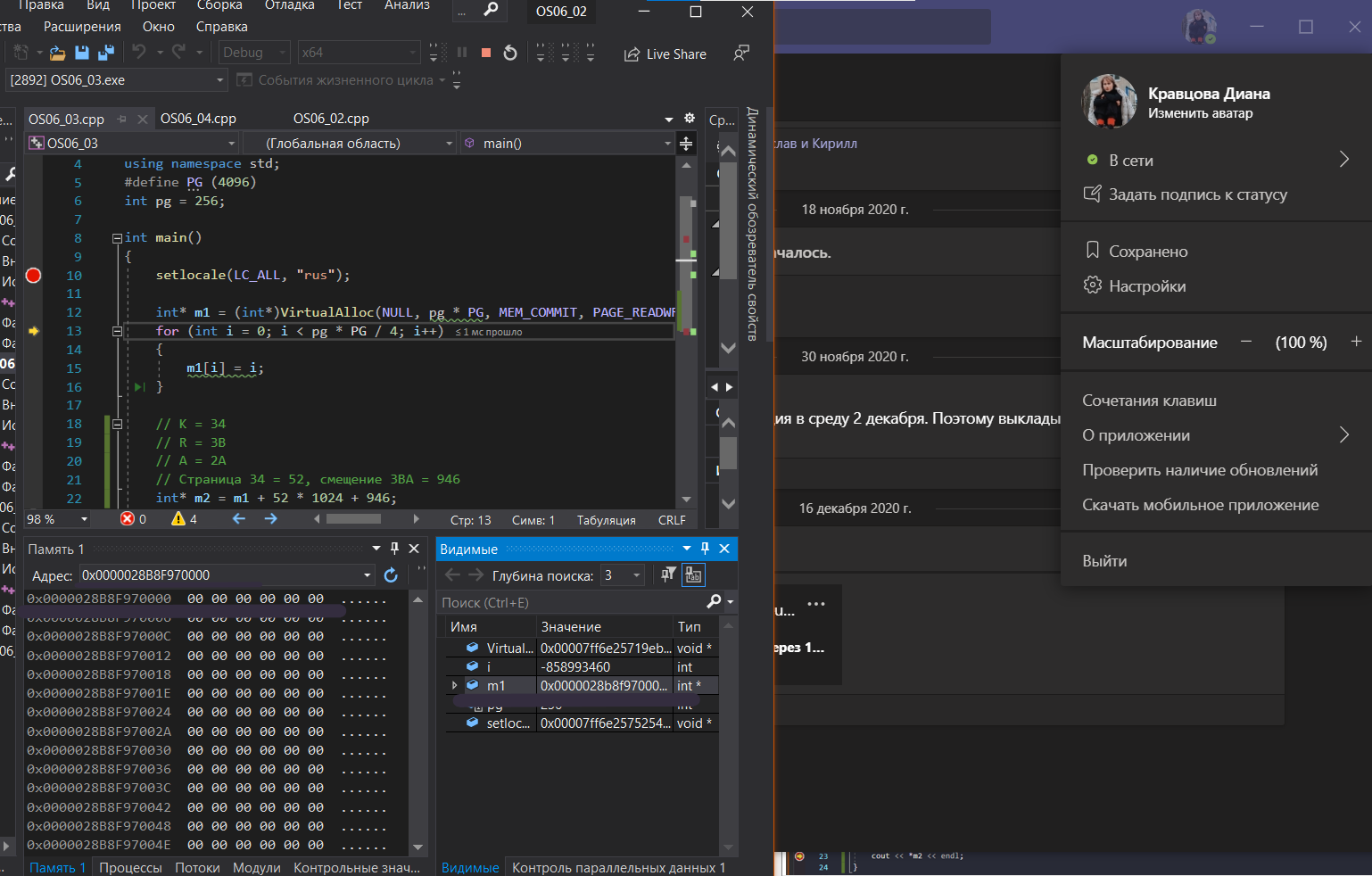
**Visual Studio не разрешает установить базовый адрес в формате, указанном в задании. Поэтому базовый адрес был установлен в позиции в алфавите, переведенные в HEX. (0xc4200000)**

**12 – C; 4 – 4; 2 – 2;**

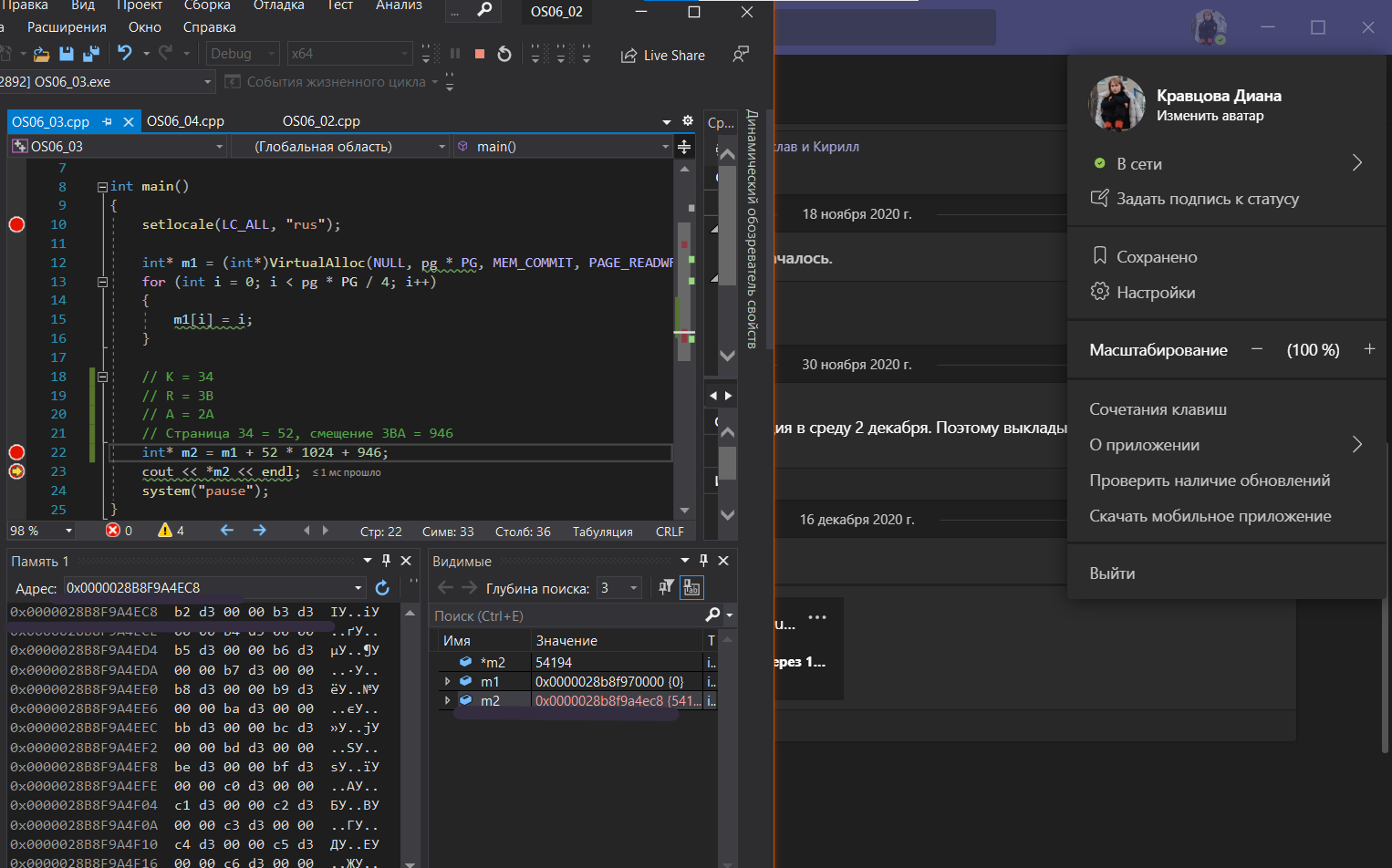


**ЗАДАНИЕ 3**

**Скриншот с выделением начального адреса выделенной памяти.**



**Скриншот с выделением смещенного адреса.**



**Расписать получение смещённого адреса.**

**Смещенный адрес получается по формуле: начальный адрес(m1) + смещение на страницу + смещение в странице.**

**Переводим первые три буквы фамилии в HEX (KRA – 343B2A);**

**Cтраница = число в нулевом байте(343B2A), т.е. 34 (52 в DEC)**

**Смещение в странице = число 12 бит в 1-ом и 2-ом байтах(343B2A), т.е. 3B2 (946 в DEC).**

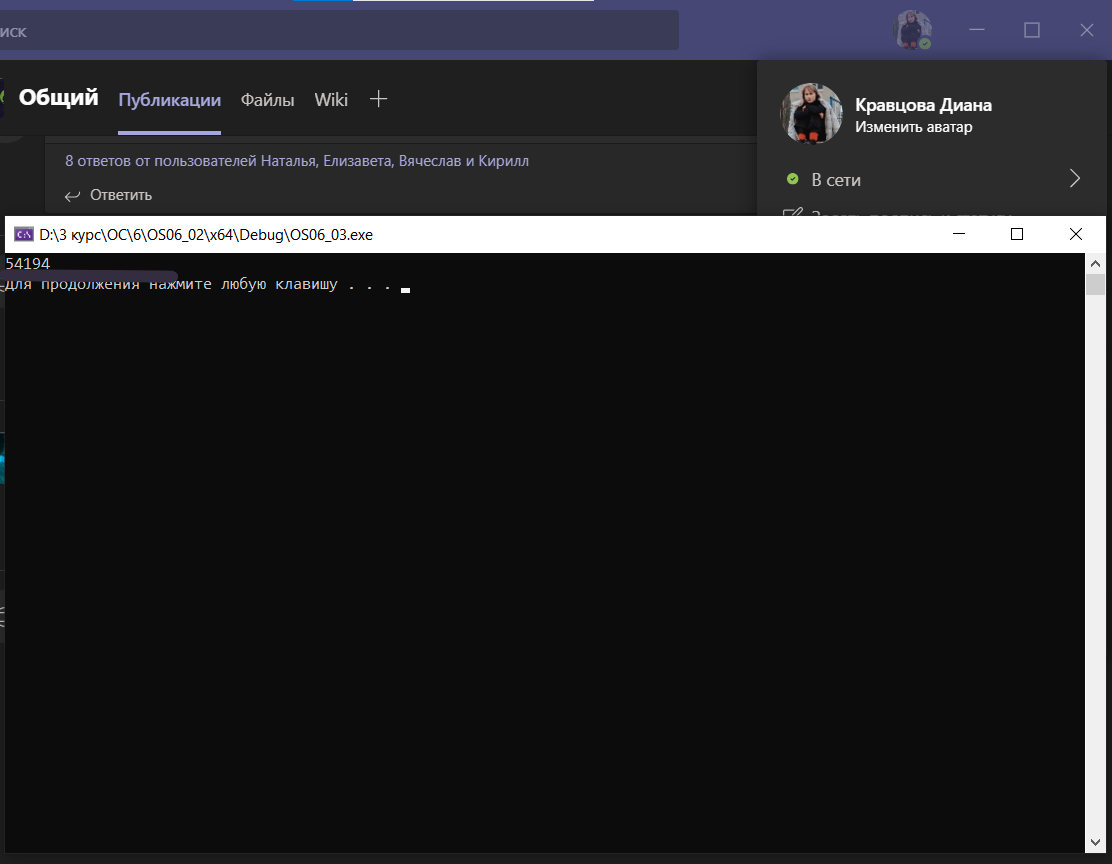
**Так как мы работаем с указателем типа int, для того чтобы перейти на 52 страницу, нам нужно 52 умножить на 1024, т.к. одна страница занимает 4096 байт, значит на одной странице помещается 1024 числа типа int, т.к. int в памяти занимает 4 байта.**

**Создаем переменную m2, которая хранит указатель типа int;**

**Адрес m2 вычисляется: начальный адрес(m1) + (номер страницы) \* (количество чисел int, помещающихся на одной странице) + смещение.**

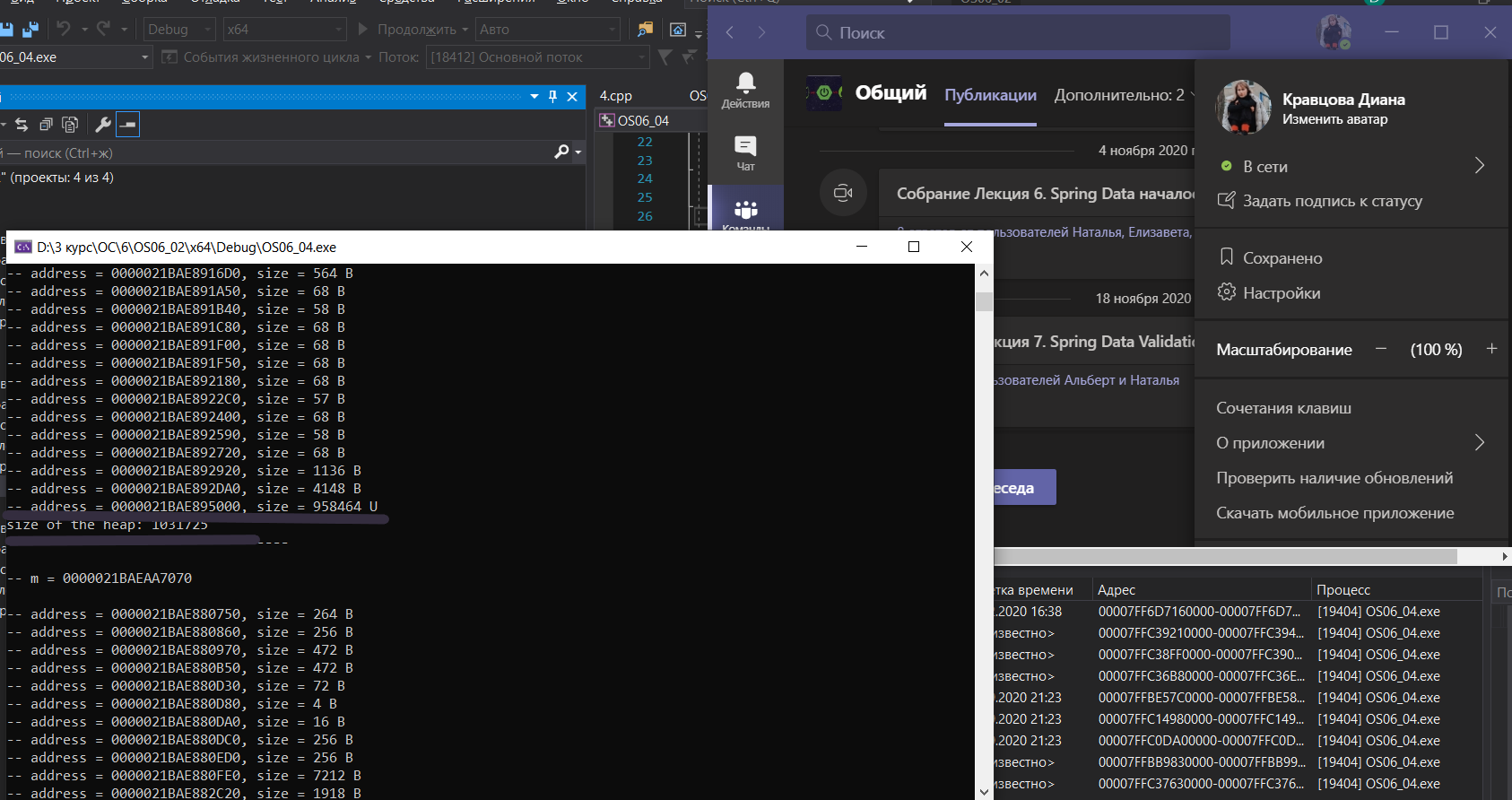
**В итоге мы сместимся на 52 страницу относительно начального адреса, а затем, прибавив 946, мы сместимся ещё на 946 чисел.**

**В памяти под этим адресом будет находиться 54194 число (52 \* 1024 + 946) массива, если первый элемент брать за 1.**

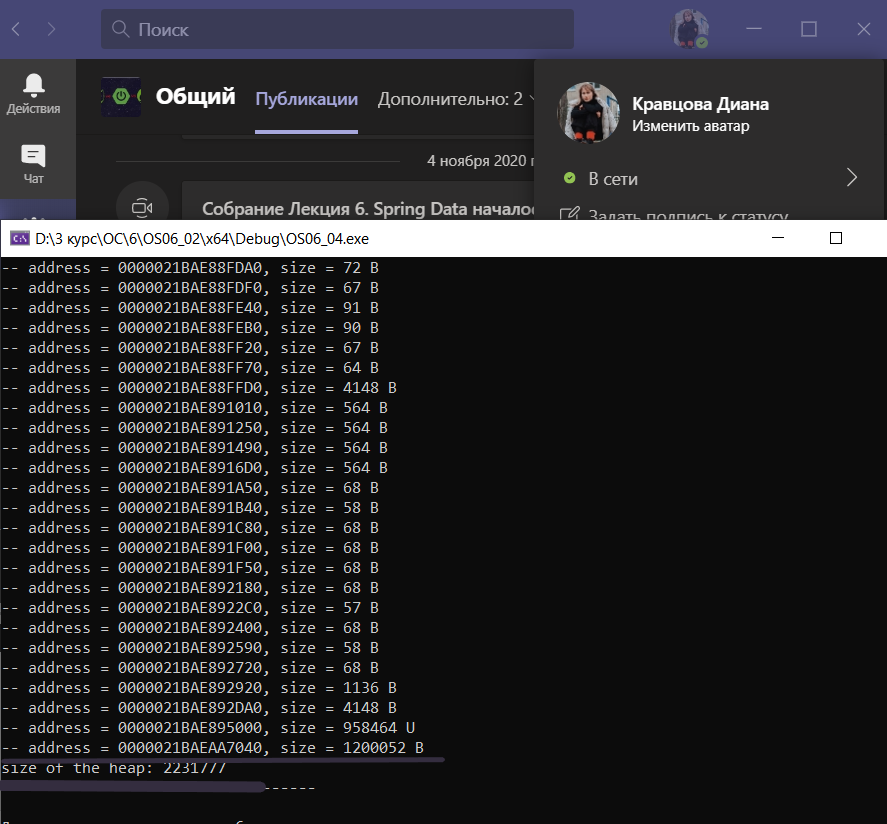


**ЗАДАНИЕ 4**

**Скриншот запуска приложения и результата работы функции sh до выделения памяти под массив.**



**Скриншот результата работы функции sh после выделения памяти под массив.**



**Используемые функции:**

**GetProcessHeap() – возвращает дескриптор к кучи по умолчанию вызывающего процесса.**

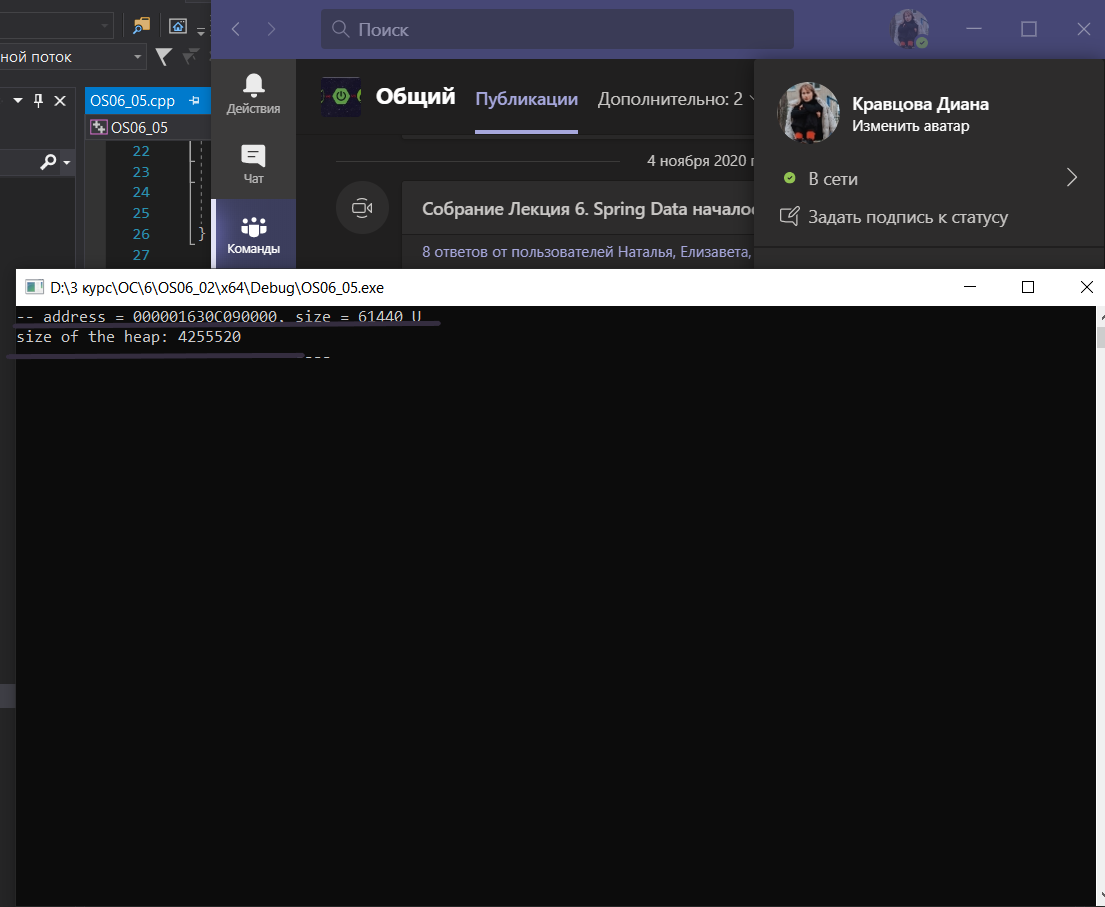
**HeapWalk(HANDLE hHeap, LPPROCESS\_HEAP\_ENTRY lpEntry), где hHeap – дескриптор кучи, lpEntry – указатель на структуру PROCESS\_HEAP\_ENTRY.**

**Функция HeapWalk используется для перемещения по памяти от узла к узлу.**

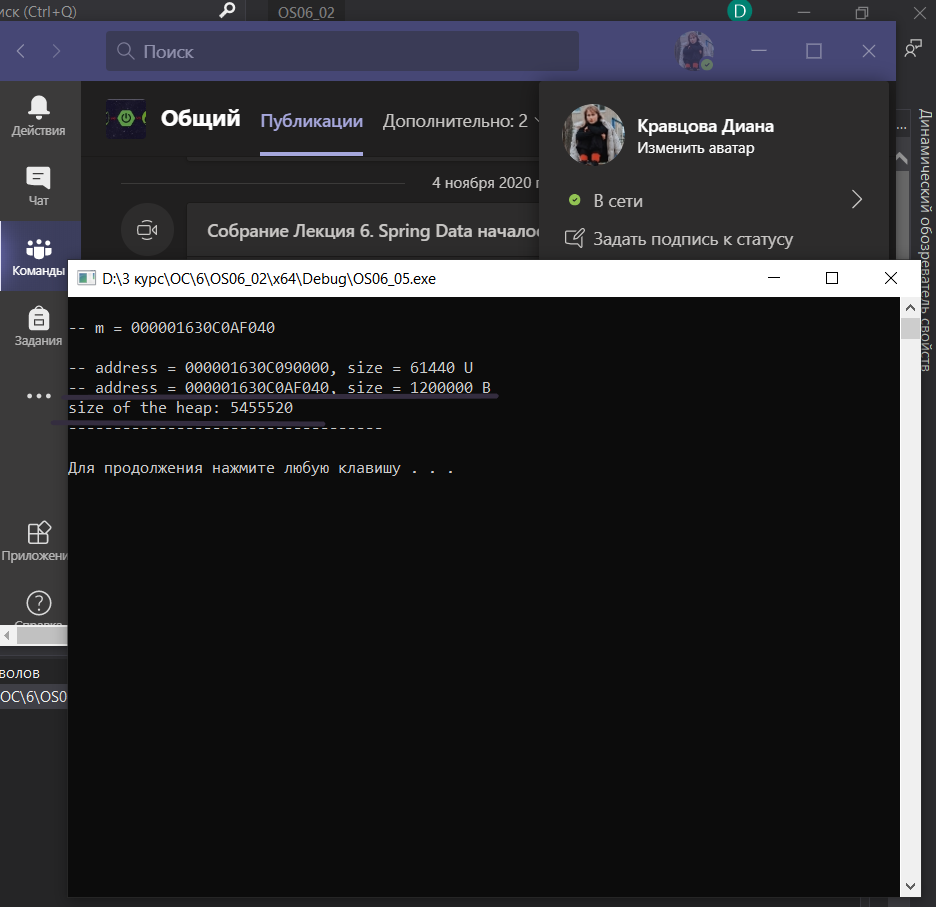
**По умолчанию куча = 1МБ (видно на первом скриншоте). Затем мы выделяем память для 300000 элементов типа int, они будут занимать 1200000 байт, т.к. число типа int занимает 4 байта.**

**На 2 скриншоте видно, что так как объема кучи по умолчанию не хватило, было выделено дополнительно 1200000 байт для хранения массива, а объем свободной памяти в куче по умолчанию остался без изменений.**

**ЗАДАНИЕ 5**

**Скриншот запуска приложения и результата работы функции sh до выделения памяти под массив. **

**Скриншот результата работы функции sh после выделения памяти под массив.**



**Используемые функции:**

**HeapCreate(DWORD flOptions, SIZE\_T dwInitializeSize, SIZE\_T dwMaximumSize), где flOptions – вариант распределения кучи, эти параметры влияют на последующий доступ к новой куче, dwInitializeSize – начальный размер кучи в байтах, dwMaximumSize – максимальны размер кучи в байтах.**

**Функция HeapCreate создает объект кучи, который может использоваться вызыващим процессом.**

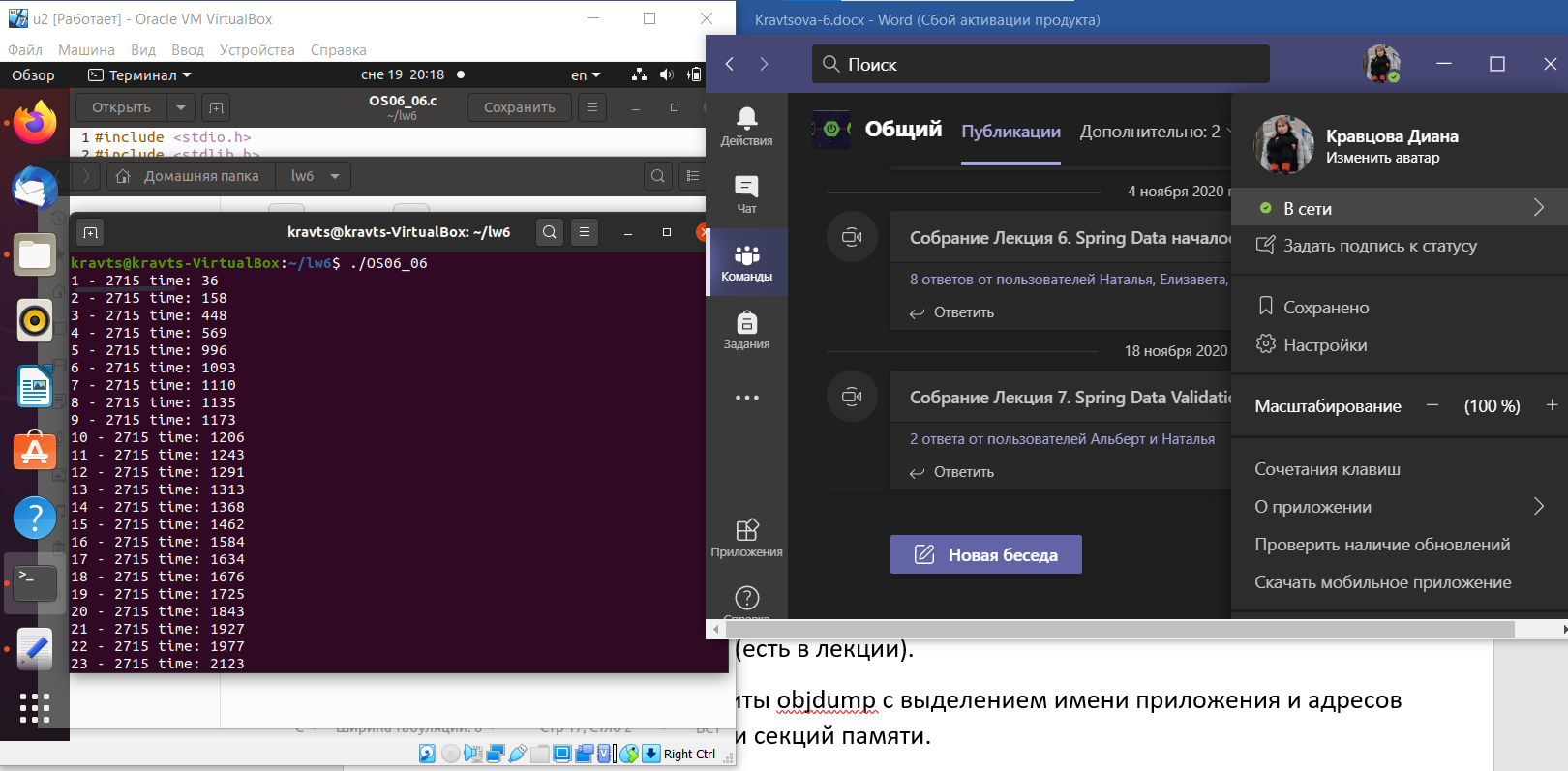
**HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE\_T dwBytes), где hHeap – дескриптор кучи, из который будет выделена память, dwFlags – варианты распределения кучи, dwBytes – количество выделяемых байтов.**

**Функция HeapAlloc выделяет блок памяти из кучи.**

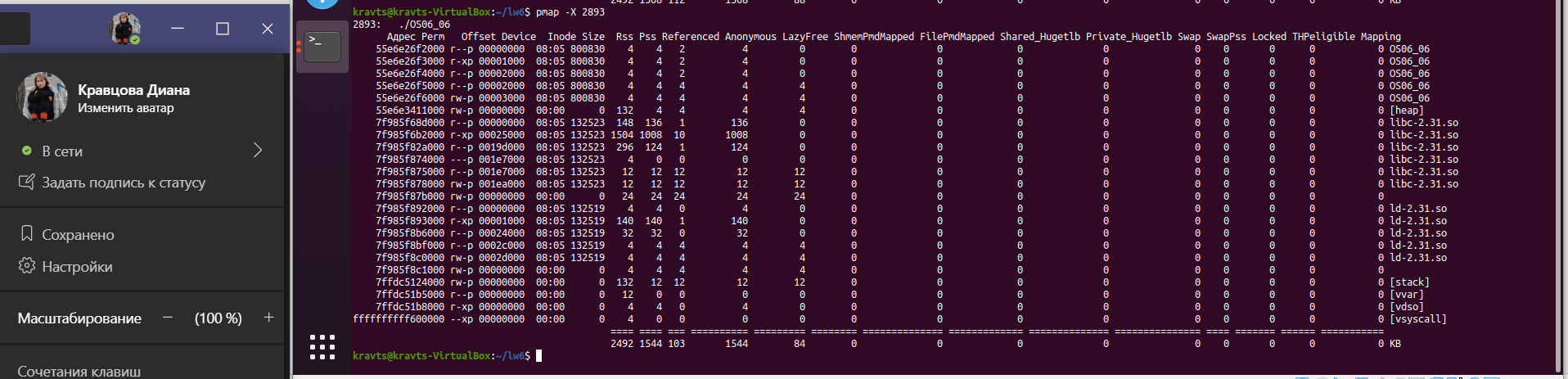
**Самый большой блок памяти, который может быть выделен из кучи, составляет чуть меньше 512 КБ для 32-битного процесса. Запросы на выделение больших блоков не выполняются, даже если максимальный размер кучи достаточно велик, чтобы вместить блок.**

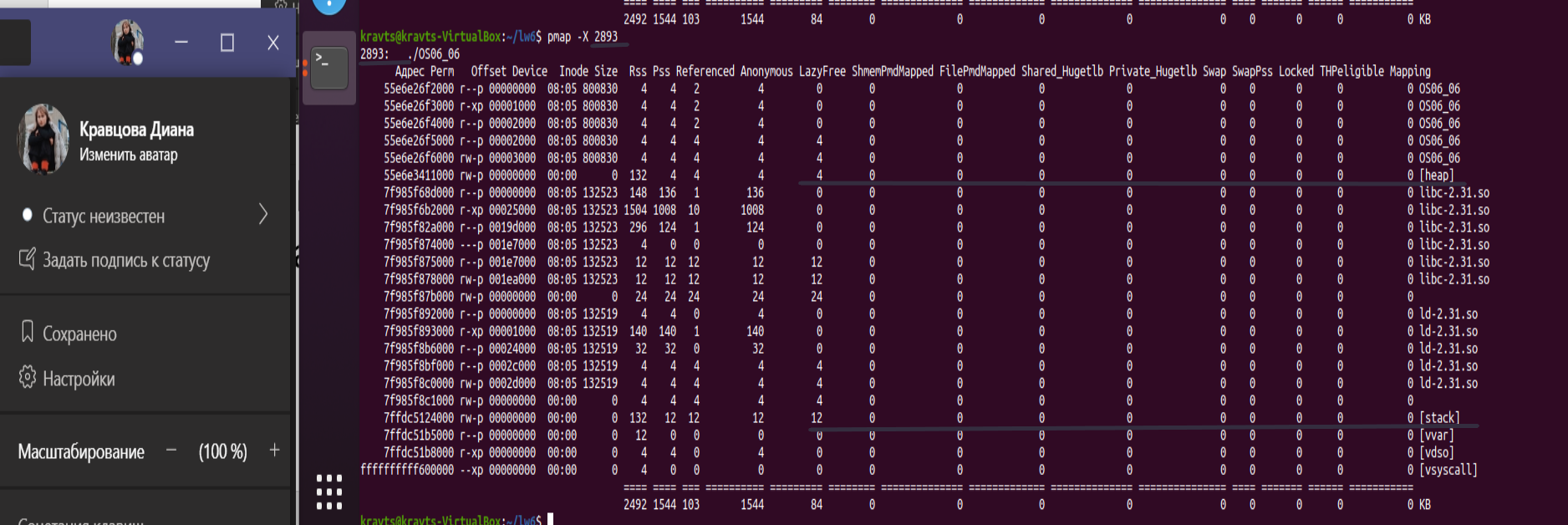
**Если dwMaximumSize равен 0, то куча может увеличиваться в размере. Размер кучи ограничен только доступной памятью. Запросы на выделение блоков памяти, превышающих лимит для кучи фиксированного размера, не завершаются автоматически; вместо этого система вызывает функцию VirtualAlloc (позволяет получить память, необходимую для больших блоков). Приложения, которым необходимо выделить большие блоки памяти, должны установить значение dwMaximumSize равным 0.**

**ЗАДАНИЕ 6**

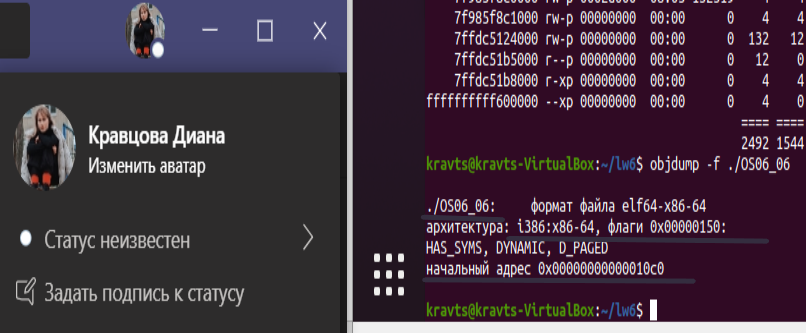
**Скриншот запуска приложения и результата хотя бы нескольких итераций с выделением PID. **

**2 скриншота для утилит с выделением PID, а также основных участков памяти для Linux-процесса (есть в лекции).**

****

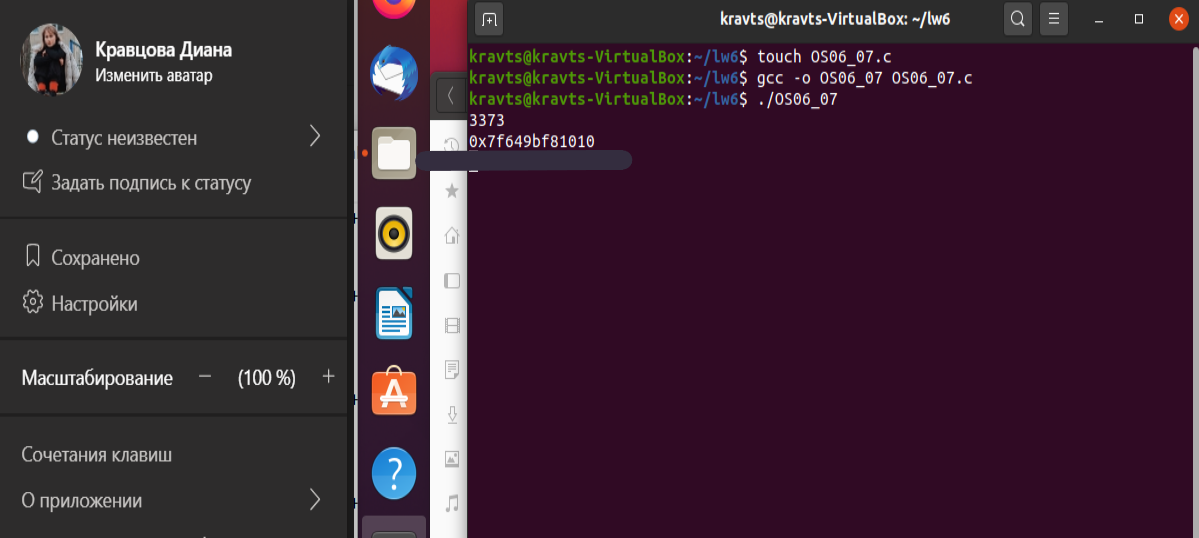


**Скриншот для утилиты objdump с выделением имени приложения и адресов указанных в условии секций памяти.**



**ЗАДАНИЕ 7**

**Скриншот запуска приложения и результата с выделением полученного адреса в памяти.**



**Скриншот утилиты с выделением адреса и имени области памяти в которой расположился выделенный массив.**

