**Лабораторная работа №2**

**Знакомство с файлами отображенными в память**

**Цель работы:** Получение практических навыков в работе с файлами отображенными в память.

**Постановка задачи для Windows:**

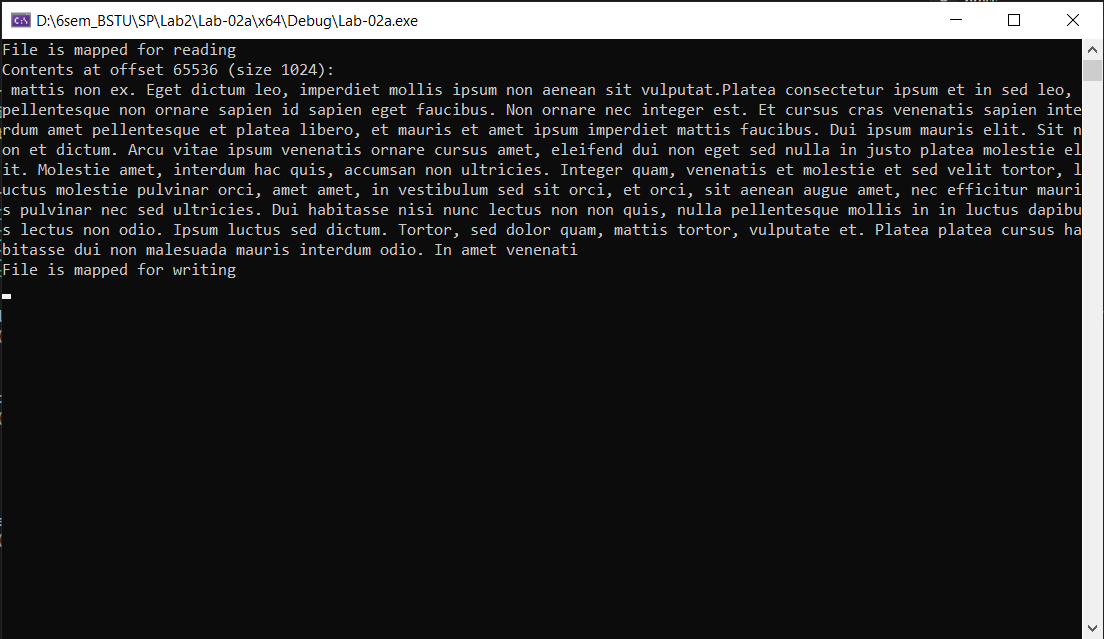
**Проект Lab-02a:**

Разработать проект который состоит из одного приложения на языке C которое содержит код который выполняет следующее:

* создаёт и использует файл отображенный в память (файл должен быть текстовый);
* создаёт представление на часть файла (должен быть задан отступ и размер не равные 0 на своё усмотрение) и выводит её содержимое на консоль;
* создаёт представление на часть файла (должен быть задан отступ и размер не равные 0 на своё усмотрение) и заполняет её нулями (\0), а также **гарантировано** **сохраняет** изменения в файл на диске (проверяется открытием файла после завершения работы программы и инспекцией содержимого);
* обрабатывает нештатные ситуации и корректно управляет ресурсами (т.е. освобождает ненужные ресурсы и т.д.).

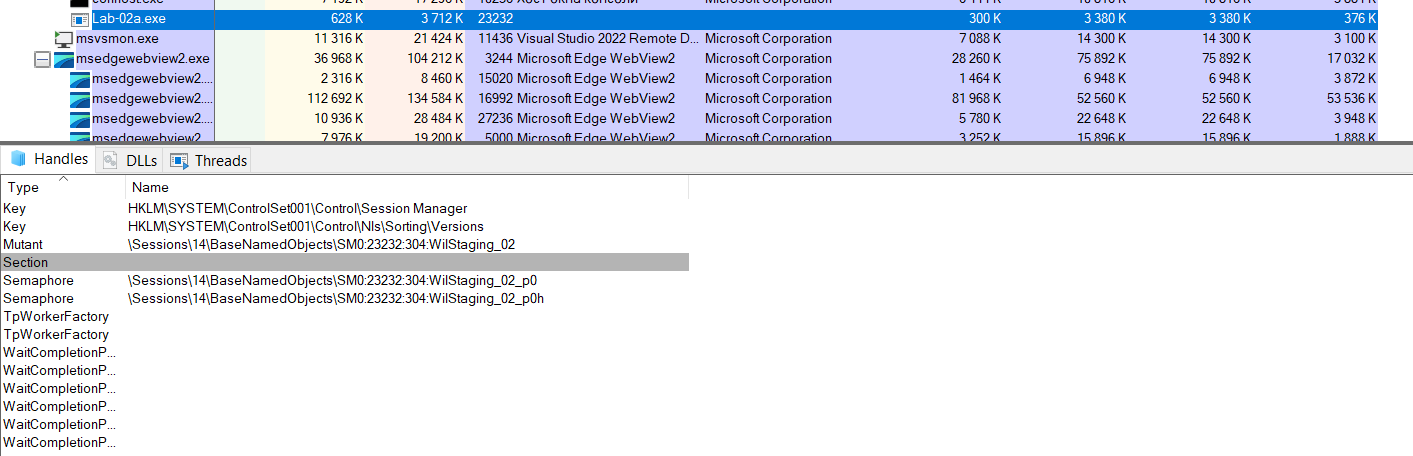
С помощью утилиты [Process Explorer](https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer) найти и изучить информацию о файле отображенном в память.

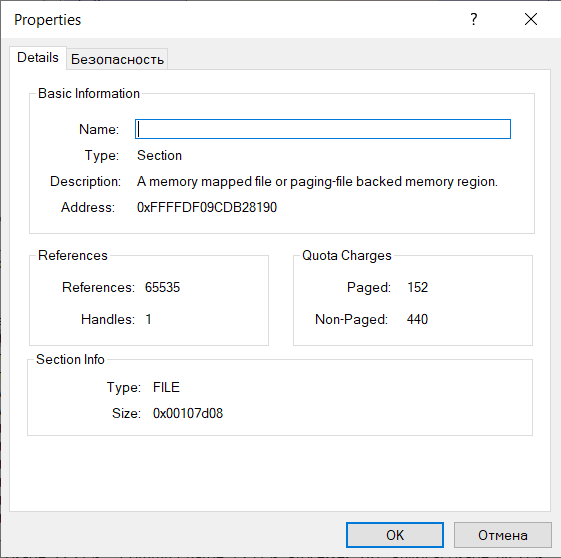
|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #define VIEW\_OFFSET 65536 // 64 \* 1024  #define VIEW\_SIZE\_1 ((SIZE\_T)1024)  #define VIEW\_SIZE\_2 ((SIZE\_T)5120) // 5 \* 1024  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  int main() {  HANDLE hFile = INVALID\_HANDLE\_VALUE;  HANDLE hMap = NULL;  LPVOID pView = NULL;  hFile = CreateFileW(L"file.txt", GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE) handle\_error("Failed to open or create file");  hMap = CreateFileMappingW(hFile, NULL, PAGE\_READWRITE, 0, 0, NULL);  if (hMap == NULL) {  CloseHandle(hFile);  handle\_error("Failed to create file mapping");  }  pView = MapViewOfFile(hMap, FILE\_MAP\_READ, 0, VIEW\_OFFSET, VIEW\_SIZE\_1);  if (pView == NULL) {  CloseHandle(hMap);  CloseHandle(hFile);  handle\_error("Failed to map view for reading");  }  printf("File is mapped for reading\n");  printf("Contents at offset %ld (size %zu):\n", (long)VIEW\_OFFSET, VIEW\_SIZE\_1);  fwrite(pView, 1, VIEW\_SIZE\_1, stdout);  printf("\n");    printf("Press enter to continue... ");  getchar();  UnmapViewOfFile(pView);  pView = MapViewOfFile(hMap, FILE\_MAP\_WRITE, 0, VIEW\_OFFSET, VIEW\_SIZE\_2);  if (pView == NULL) {  CloseHandle(hMap);  CloseHandle(hFile);  handle\_error("Failed to map view for writing");  }  printf("File is mapped for writing\n");  memset(pView, '0', VIEW\_SIZE\_2);  printf("Press enter to continue... ");  getchar();  FlushViewOfFile(pView, VIEW\_SIZE\_2);  UnmapViewOfFile(pView);  CloseHandle(hMap);  CloseHandle(hFile);  printf("File mapping is ended\n");  printf("Press enter to continue... ");  getchar();  return 0;  } |

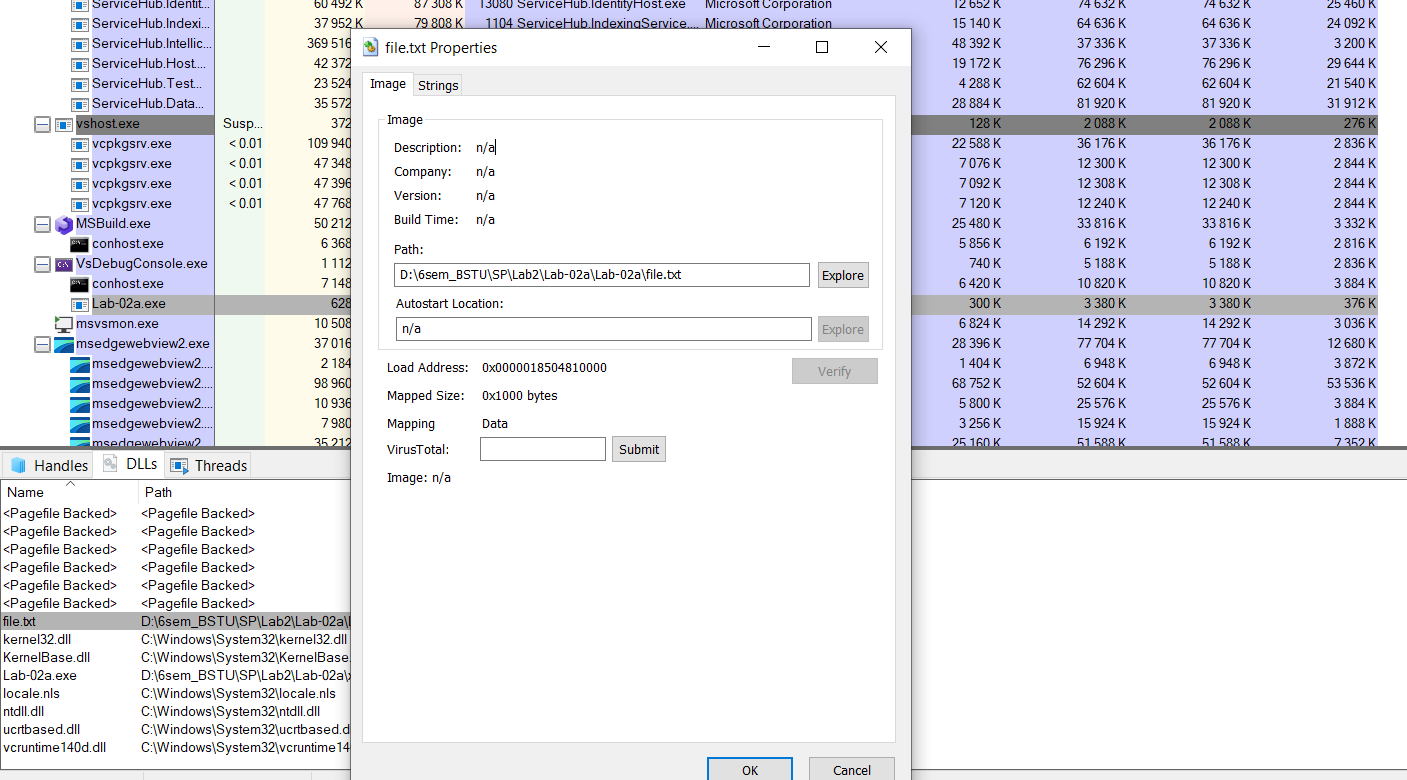


Дескриптор типа Section представляет собой объект, связанный с файлом, отображённым в память или областью памяти, поддерживаемой страничным файлом.

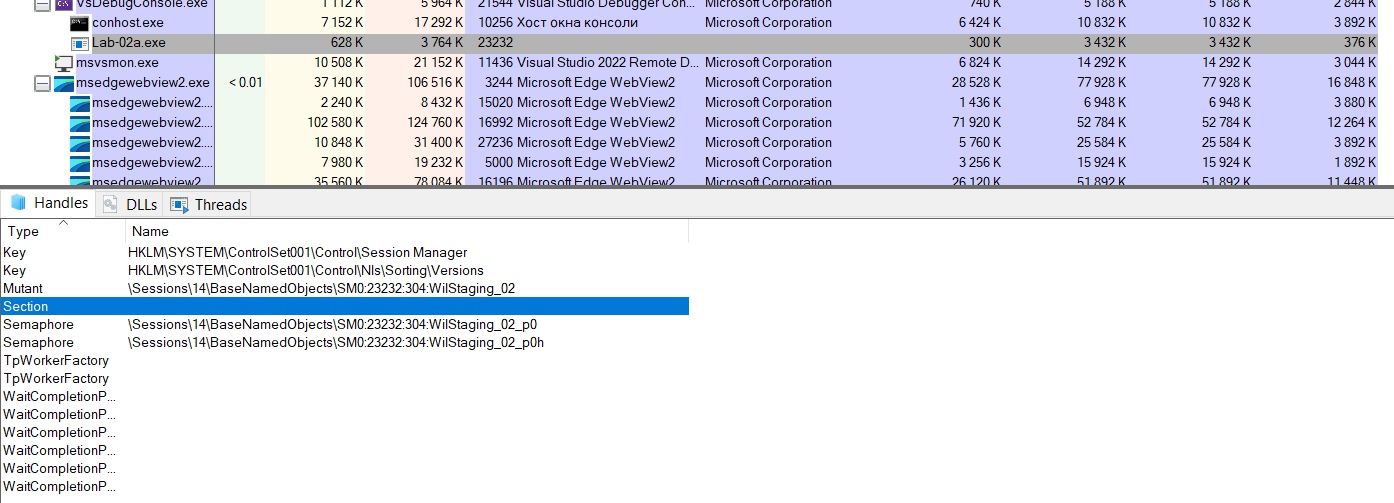
Первый MapViewOfFile:

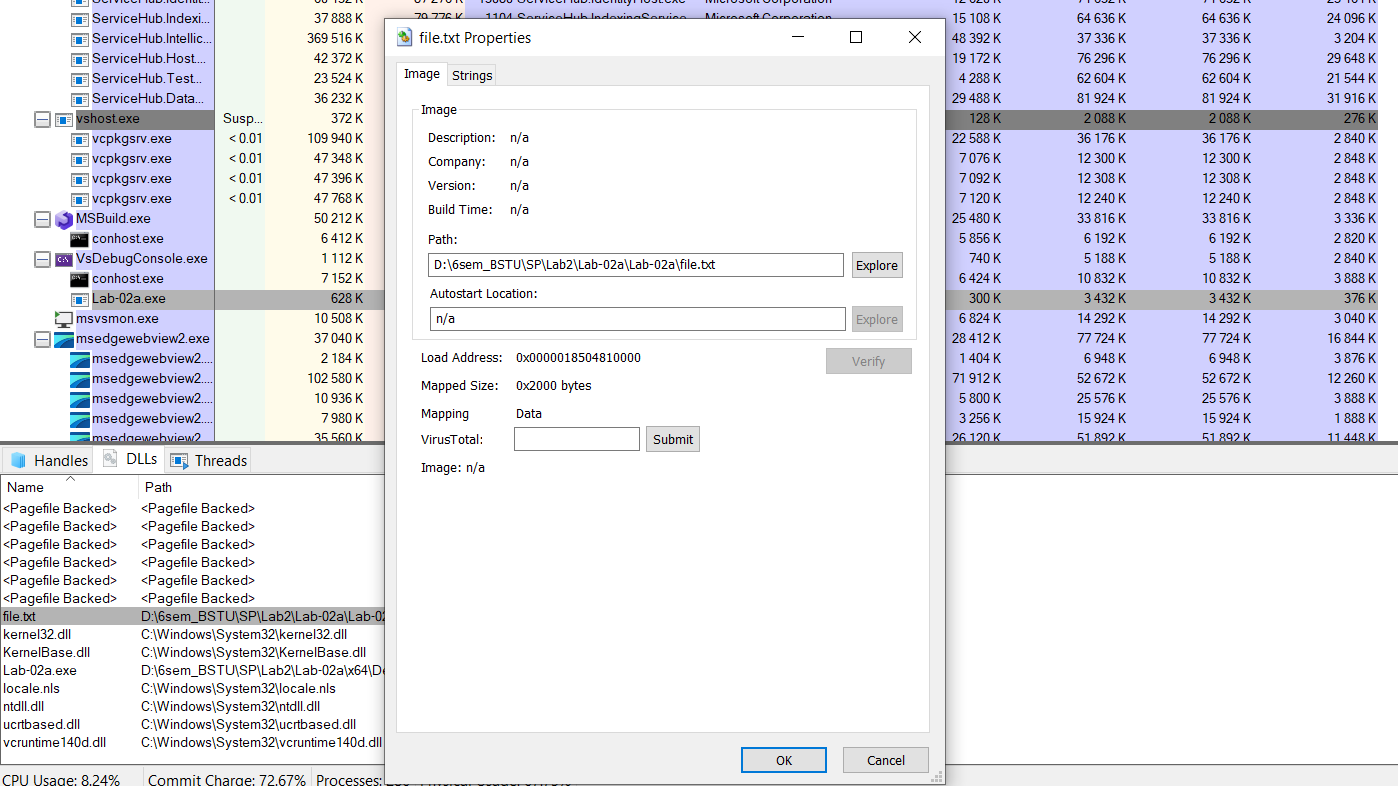




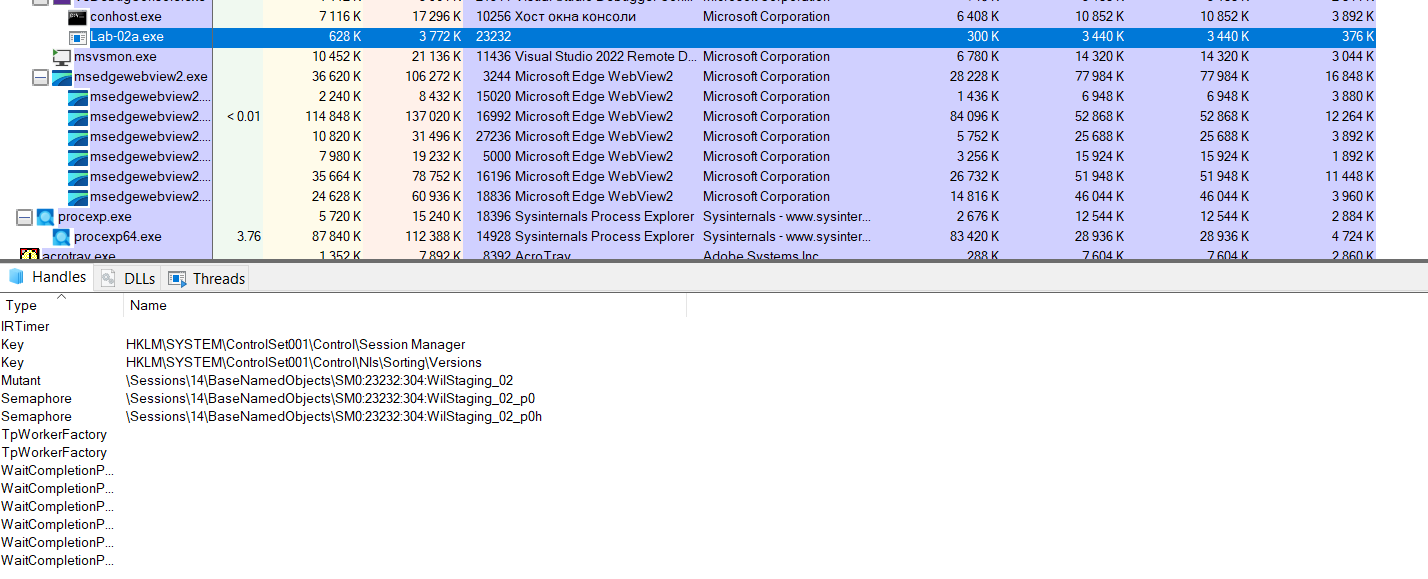


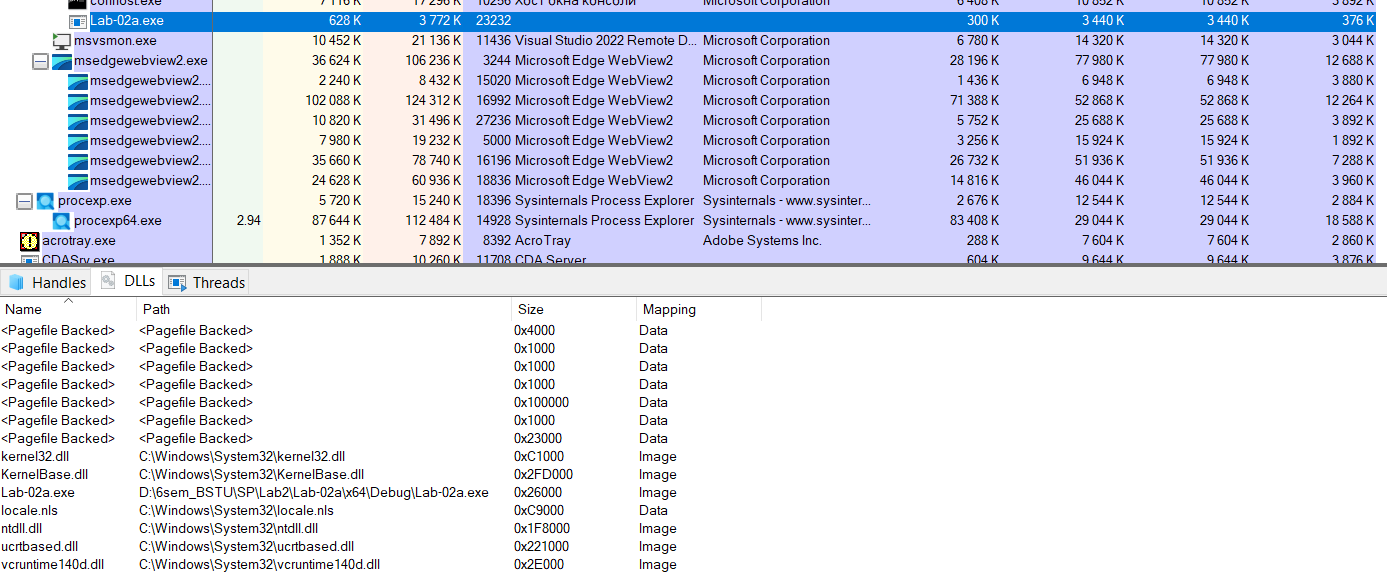
Второй MapViewOfFile:





После UnmapViewOfFile:





Size 0x1000: это размер сопоставленной области в шестнадцатеричном формате. 0x1000 равно 4096 байтам. Это размер отображаемой области памяти, выделенной для file.txt. Это минимальный размер, который может быть сопоставлен. Операционная система обычно выделяет память блоками по 4 КБ (размер страницы памяти). Даже если мы сопоставляем меньший объем, например 64 байта, операционная система может выделить целую страницу (4 КБ).

Size 0x2000 равно 8192 байта (8 кб = 2 страницы).

**Проект Lab-02b:**

Разработать проект который состоит из одного приложения на языке C которое содержит код который выполняет следующее:

* создаёт и использует файл отображенный в память (отображаемый файл должен быть текстовый и размером не менее 512 МиБ. Файл можно сгенерировать [тут](https://testdatahub.com/generate_files)). Объект отображения должен создаваться на весь файл целиком;
* создаёт представление на часть файла размером не менее 128 МиБ и выводит её содержимое на консоль;
* освобождает все ресурсы и завершает свою работу.

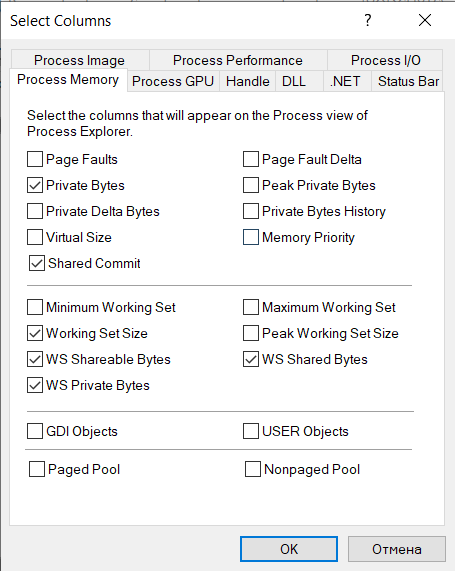
С помощью утилиты [Process Explorer](https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer) найти и изучить информацию о том, в какой момент времени процесс начинает потреблять больше всего физической памяти. (Например, сделайте паузы между этапами с помощью getchar)

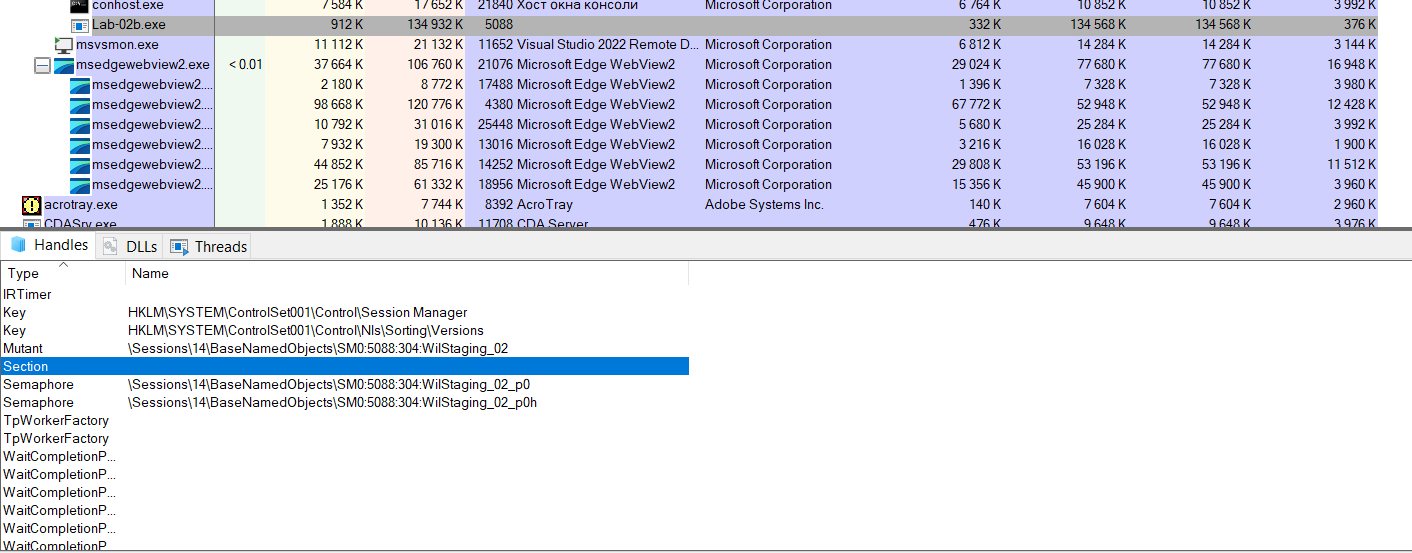
Информация: для изучения объёмов памяти убедитесь, что у вас отображаются необходимые столбцы (Меню View -> Select Columns-> Process Memory. Из необходимых: Private Bytes, Shared Commit, Working Set Size, и всё что начинается с WS-).

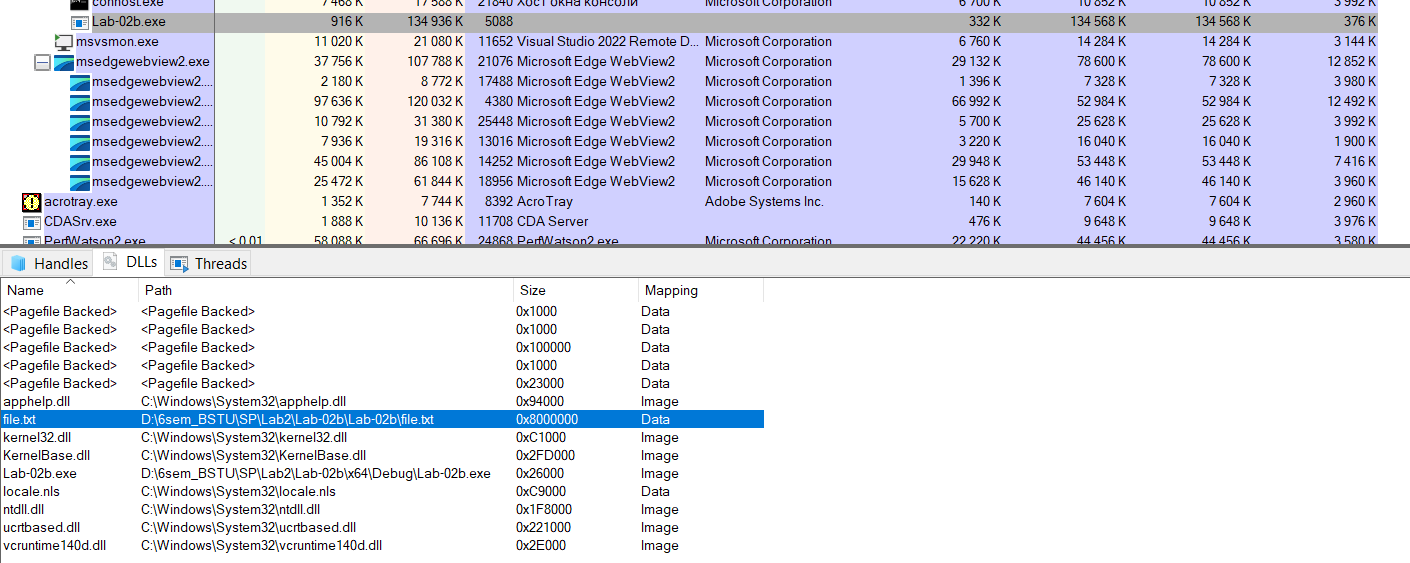
***Вопрос:*** *Поясните**полученный результат исследования в утилите Process Explorer. Данные были в Private или Shareable памяти?*

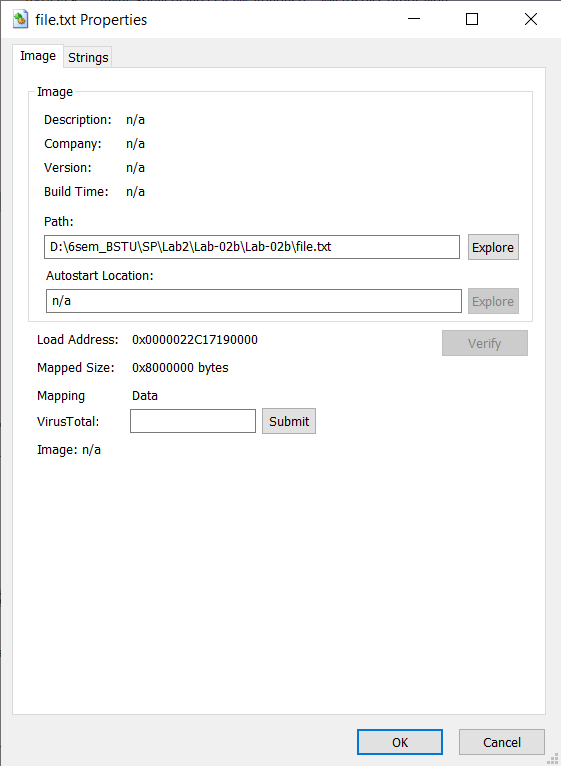
|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #define VIEW\_OFFSET 0  #define VIEW\_SIZE 134217728 // 128 \* 1024 \* 1024  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  int main() {  HANDLE hFile = INVALID\_HANDLE\_VALUE;  HANDLE hMap = NULL;  LPVOID pView = NULL;  hFile = CreateFileW(L"text.txt", GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE) handle\_error("Failed to open file");  hMap = CreateFileMappingW(hFile, NULL, PAGE\_READWRITE, 0, 0, NULL);  if (hMap == NULL) {  CloseHandle(hFile);  handle\_error("Failed to create file mapping");  }  pView = MapViewOfFile(hMap, FILE\_MAP\_READ, 0, VIEW\_OFFSET, VIEW\_SIZE);  if (pView == NULL) {  CloseHandle(hMap);  CloseHandle(hFile);  handle\_error("Failed to map view for reading");  }  printf("File is mapped for reading\n");  printf("Contents at offset %d (size %u):\n", VIEW\_OFFSET, VIEW\_SIZE);  fwrite(pView, 1, VIEW\_SIZE, stdout);  printf("\n");  printf("Press enter to continue... ");  getchar();  UnmapViewOfFile(pView);  CloseHandle(hMap);  CloseHandle(hFile);  printf("File mapping is ended\n");  return 0;  } |

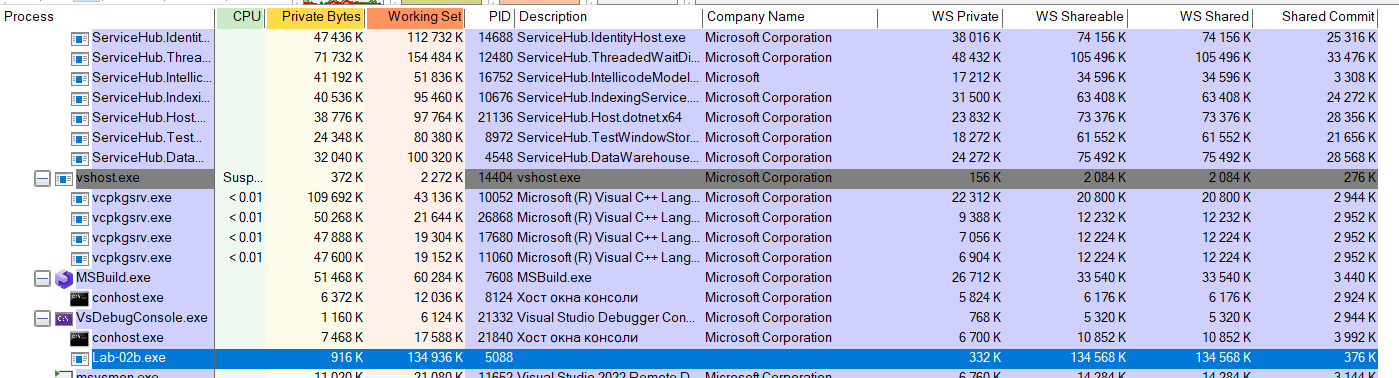
**

**

**

**

**

**

**Виртуальная память, подтверждённая процессом (Shared Commit + Private Bytes)**– это полный объём адресного пространства, который операционная система выделила процессу.

**Рабочий набор (working set)** – это та часть виртуальной памяти процесса, которая в данный момент находится в физической оперативной памяти (RAM). Это те страницы виртуальной памяти, которые процесс активно использует.

**Private память** – это те страницы, которые используются исключительно данным процессом. Обычно сюда относятся данные, выделенные для стека, кучи и внутренних структур данных.

**Shareable память** – это те страницы, которые могут одновременно использоваться несколькими процессами. Обычно сюда входят страницы, полученные через memory-mapped файлы – например, разделяемые библиотеки (DLL) или другие файлы, отображённые в память. Такие страницы могут использоваться разными процессами, что позволяет экономить физическую память за счёт разделения общих данных.

**WS Shareable** – это объём физических страниц в рабочем наборе, которые могут быть разделены (например, страницы из DLL или memory‑mapped файлов). **WS Shared** – это та часть WS Shareable, которая в данный момент фактически используется несколькими процессами одновременно.

* Очень низкий объём Private Bytes и WS Private означает, что процесс почти не использует свою личную память.
* Большая часть рабочего набора приходится на разделяемую память. Такое распределение типично для страниц, отображённых из файлов.

WS > (Shared Commit + Private Bytes) – процесс может создать зарезервированное отображение файла большого размера, но фактически использовать (и, следовательно, загрузить в RAM) только небольшую его часть. Эта небольшая часть войдёт в WS, но весь размер зарезервированного отображения не будет сразу же учтён ни в Shared Commit, ни в Private Bytes (потому что память под него ещё не выделена – не закоммичена). Память будет "коммититься" (и учитываться в Shared Commit или Private Bytes) по мере обращения к разным частям файла.

**Проект Lab-02c:**

Разработать проект который состоит из двух приложений на языке С.

**Первое приложение должно**:

* создавать отображение на базе страничного файла с именем «Lab-02»;
* создавать представление размером ровно 64 КиБ с отступом 0;
* в цикле на 10 итераций записывать массив целых чисел размером 640 КиБ (по 64 КиБ на каждой итерации);
* освобождать все ресурсы и завершать свою работу.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <windows.h>  #define FILE\_MAPPING\_NAME L"Lab-02"  #define VIEW\_SIZE (SIZE\_T)65536 // 64 \* 1024  #define ITERATIONS 10  #define ARRAY\_SIZE 655360 // 640 \* 1024  #define INT\_ARRAY\_SIZE (ARRAY\_SIZE / sizeof(int)) // Number of ints in the array  #define MUTEX\_NAME L"Lab-02\_Mutex"  #define DATA\_READY\_EVENT\_NAME L"Lab-02\_DataReady"  #define DATA\_READ\_EVENT\_NAME L"Lab-02\_DataRead"  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  int main() {  HANDLE hFileMapping;  LPVOID pView;  HANDLE hMutex;  HANDLE hDataReadyEvent;  HANDLE hDataReadEvent;  int\* data;  hFileMapping = CreateFileMappingW(  INVALID\_HANDLE\_VALUE,  NULL,  PAGE\_READWRITE,  0,  ARRAY\_SIZE,  FILE\_MAPPING\_NAME  );  if (hFileMapping == NULL) {  handle\_error("Failed to create file mapping");  }  hMutex = CreateMutexW(  NULL,  FALSE,  MUTEX\_NAME  );  if (hMutex == NULL) {  CloseHandle(hFileMapping);  handle\_error("Could not create mutex");  }  hDataReadyEvent = CreateEventW(NULL, FALSE, FALSE, DATA\_READY\_EVENT\_NAME);  if (hDataReadyEvent == NULL) {  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Could not create DataReady event");  }  hDataReadEvent = CreateEventW(NULL, FALSE, FALSE, DATA\_READ\_EVENT\_NAME);  if (hDataReadEvent == NULL) {  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Could not create DataRead event");  }  int\* write\_data = (int\*)malloc(ARRAY\_SIZE);  if (write\_data == NULL) {  CloseHandle(hDataReadEvent);  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Could not allocate memory for data to write");  }  for (int i = 0; i < INT\_ARRAY\_SIZE; ++i) {  write\_data[i] = i;  }  pView = MapViewOfFile(  hFileMapping,  FILE\_MAP\_ALL\_ACCESS,  0,  0,  VIEW\_SIZE  );  if (pView == NULL) {  CloseHandle(hDataReadEvent);  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Failed to map view for reading");  }  for (int i = 0; i < ITERATIONS; ++i) {  if (i > 0) {  WaitForSingleObject(hDataReadEvent, INFINITE);  }  WaitForSingleObject(hMutex, INFINITE);  data = (int\*)pView;  memcpy(data, write\_data + (i \* (VIEW\_SIZE / sizeof(int))), VIEW\_SIZE);  printf("Writer: Wrote iteration %d\n", i + 1);  ReleaseMutex(hMutex);  SetEvent(hDataReadyEvent);  }  UnmapViewOfFile(pView);  free(write\_data);  CloseHandle(hDataReadEvent);  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  printf("Writer: Finished writing. Press Enter to release resources\n");  getchar();  printf("Writer: Resources released. Exiting\n");  return 0;  } |

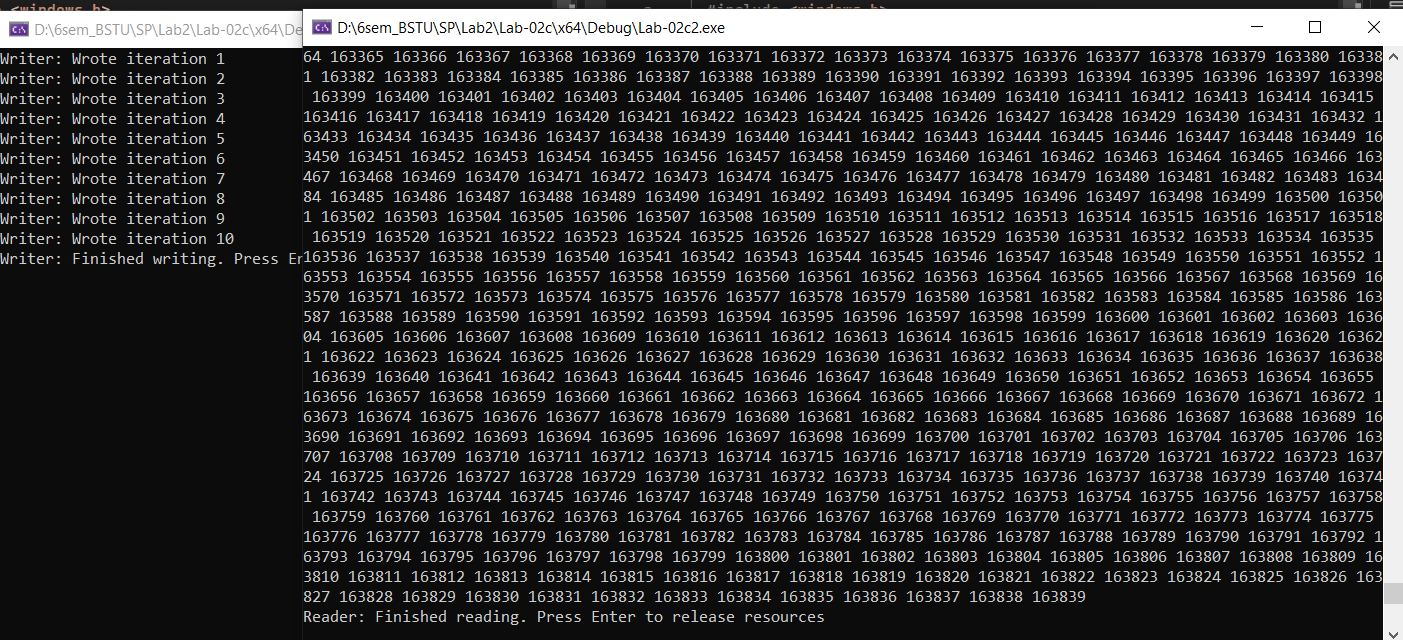
**Второе приложение должно**:

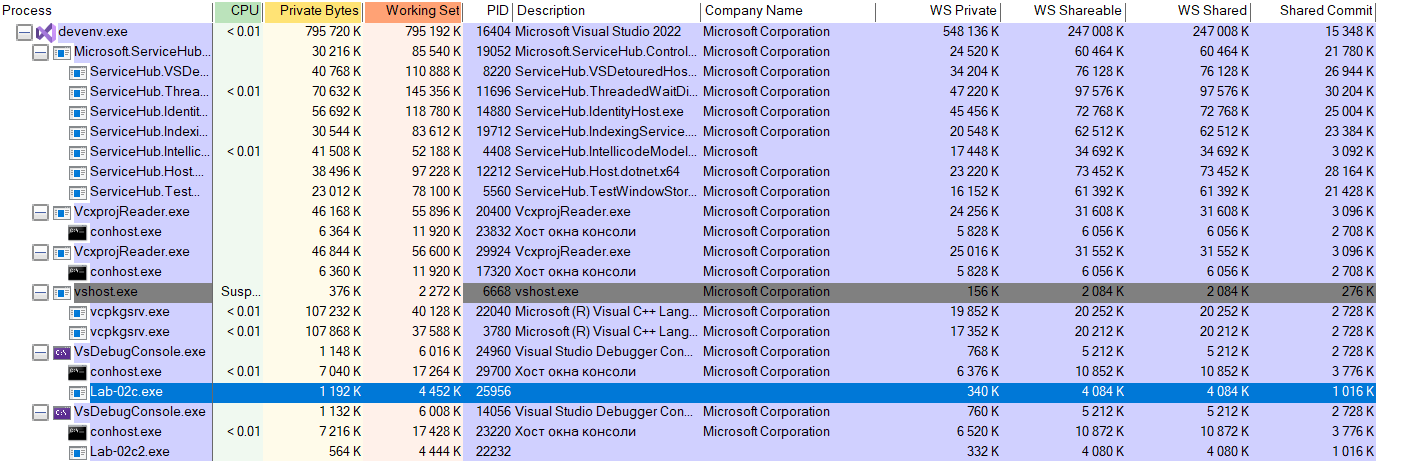
* открывать существующее отображение на базе страничного файла с именем «Lab-02»;
* создавать представление размером ровно 64 КиБ с отступом 0;
* в цикле на 10 итераций считывать массив целых чисел размером 640 КиБ (по 64 КиБ на каждой итерации) и выводить на консоль;
* освобождать все ресурсы и завершать свою работу.

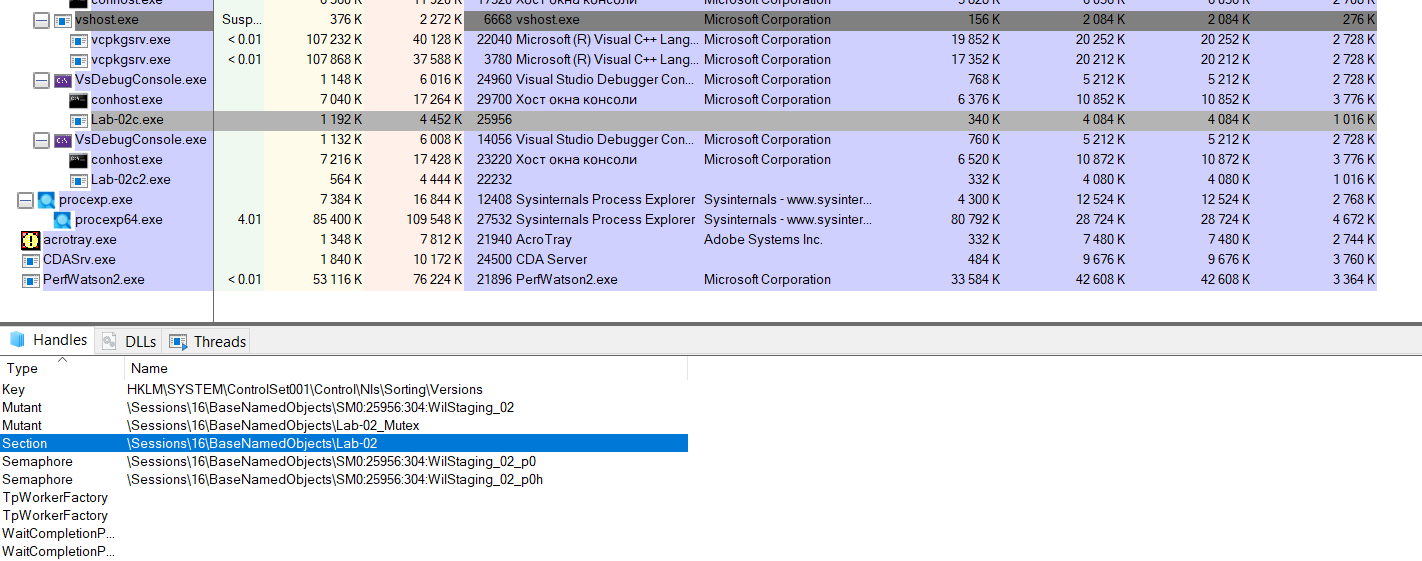
Работа приложений в циклах должна быть синхронизирована с использованием «Мьютекса».

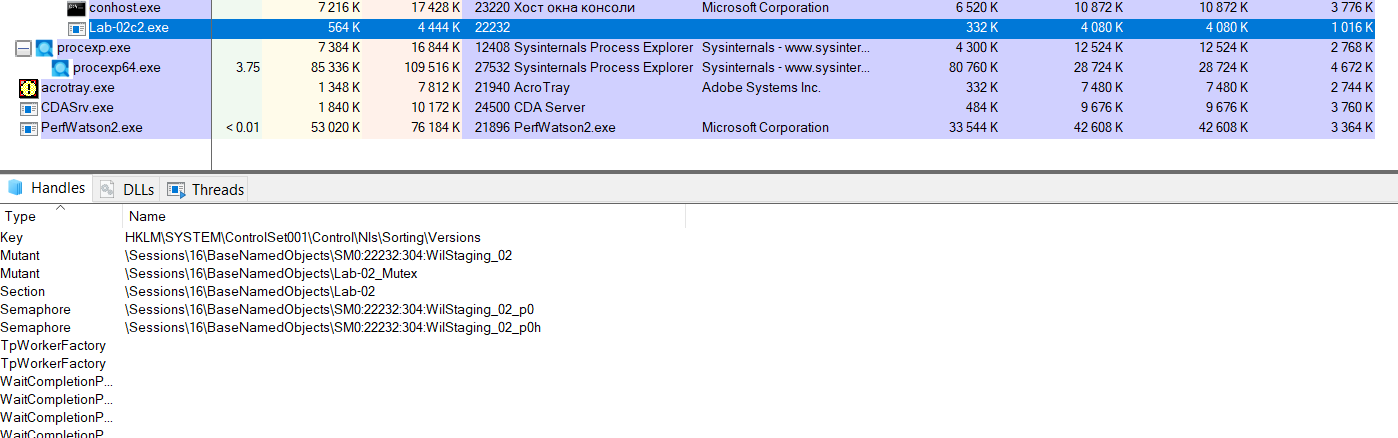
С помощью утилиты [Process Explorer](https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer) найти и изучить информацию об отображеннии на основе страничных файлов. (Например, сделайте паузы между этапами с помощью getchar)

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <windows.h>  #define FILE\_MAPPING\_NAME L"Lab-02"  #define VIEW\_SIZE (SIZE\_T)65536 // 64 \* 1024  #define ITERATIONS 10  #define ARRAY\_SIZE 655360 // 640 \* 1024  #define INT\_ARRAY\_SIZE (ARRAY\_SIZE / sizeof(int))  #define MUTEX\_NAME L"Lab-02\_Mutex"  #define DATA\_READY\_EVENT\_NAME L"Lab-02\_DataReady"  #define DATA\_READ\_EVENT\_NAME L"Lab-02\_DataRead"  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  int main() {  HANDLE hFileMapping;  LPVOID pView;  HANDLE hMutex;  HANDLE hDataReadyEvent;  HANDLE hDataReadEvent;  int\* data;  hFileMapping = OpenFileMappingW(FILE\_MAP\_ALL\_ACCESS, FALSE, FILE\_MAPPING\_NAME);  if (hFileMapping == NULL) {  handle\_error("Failed to open file mapping");  }  hMutex = OpenMutexW(SYNCHRONIZE, FALSE, MUTEX\_NAME);  if (hMutex == NULL) {  CloseHandle(hFileMapping);  handle\_error("Could not open mutex");  }  hDataReadyEvent = OpenEventW(EVENT\_MODIFY\_STATE | SYNCHRONIZE, FALSE, DATA\_READY\_EVENT\_NAME);  if (hDataReadyEvent == NULL) {  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Could not open DataReady event");  }  hDataReadEvent = OpenEventW(EVENT\_MODIFY\_STATE | SYNCHRONIZE, FALSE, DATA\_READ\_EVENT\_NAME);  if (hDataReadEvent == NULL) {  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Could not open DataRead event");  }  pView = MapViewOfFile(hFileMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, VIEW\_SIZE);  if (pView == NULL) {  CloseHandle(hDataReadEvent);  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  handle\_error("Failed to map view for reading");  }  data = (int\*)pView;  for (int i = 0; i < ITERATIONS; ++i) {  WaitForSingleObject(hDataReadyEvent, INFINITE);  WaitForSingleObject(hMutex, INFINITE);  printf("Reader: Reading iteration %d\n", i + 1);  for (int j = 0; j < VIEW\_SIZE / sizeof(int); ++j) {  printf("%d ", data[j]);  }  ReleaseMutex(hMutex);  SetEvent(hDataReadEvent);  }  UnmapViewOfFile(pView);  CloseHandle(hDataReadEvent);  CloseHandle(hDataReadyEvent);  CloseHandle(hMutex);  CloseHandle(hFileMapping);  printf("Reader: Finished reading. Press Enter to release resources\n");  getchar();  printf("Reader: Resources released. Exiting\n");  return 0;  } |









Именованные объекты Section имеют уникальные имена и могут использоваться для межпроцессного взаимодействия, позволяя нескольким процессам совместно использовать одну и ту же область памяти.

**Постановка задачи для Linux:**

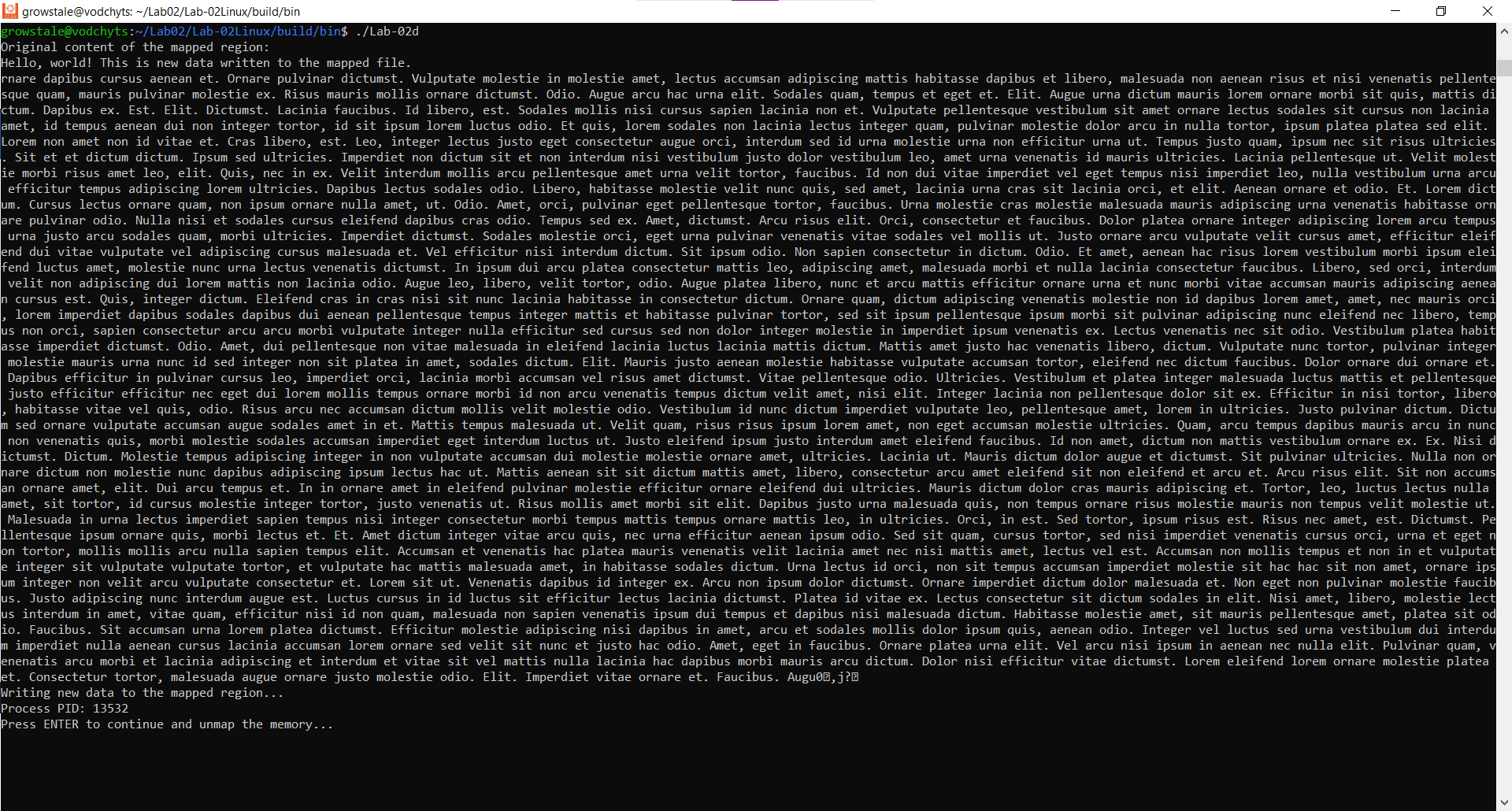
**Проект Lab-02d:**

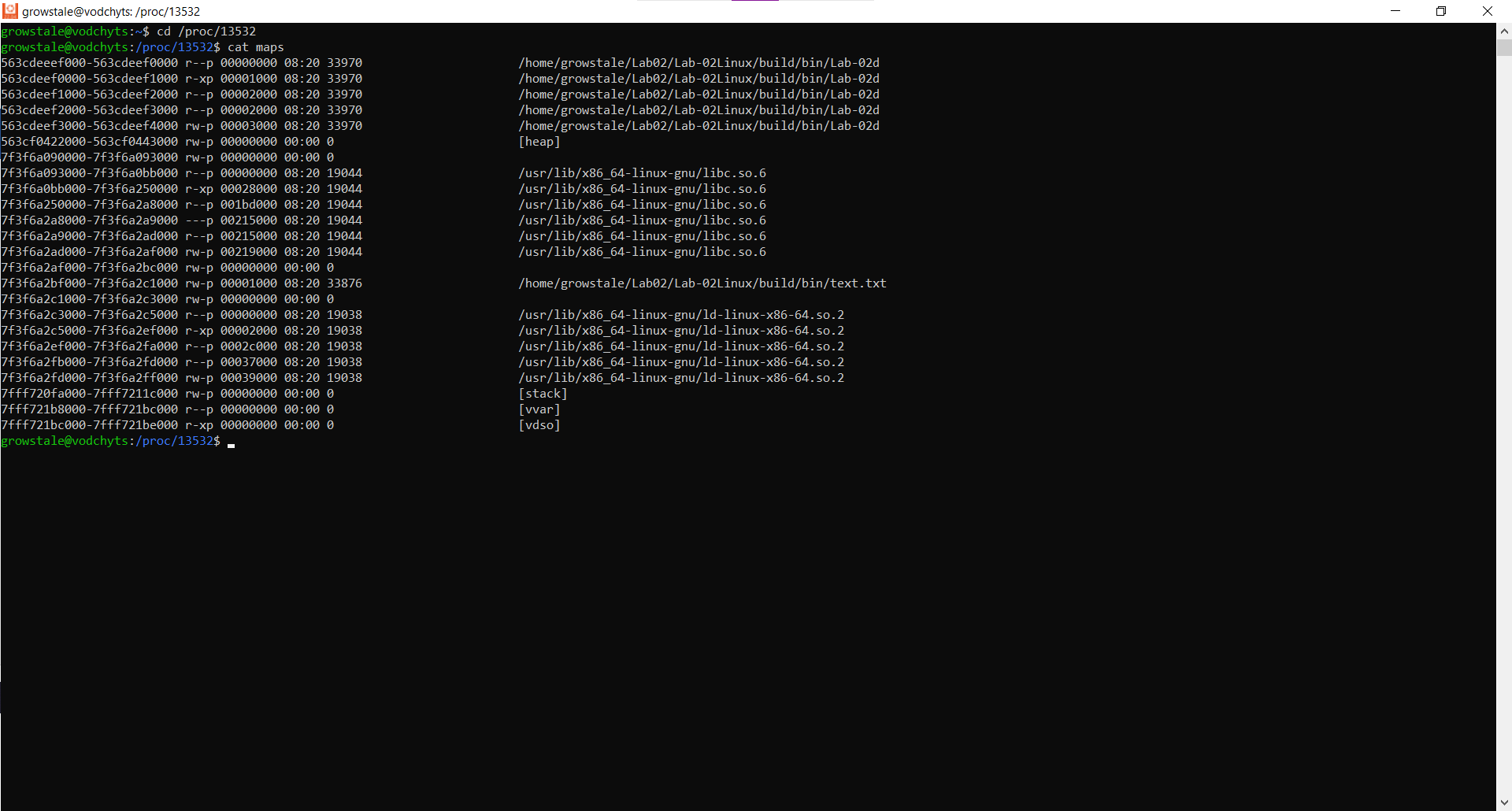
Разработать проект который состоит из одного приложения на языке C которое содержит код который выполняет следующее:

* создаёт приватное файловое отображение (файл должен быть текстовый) произвольного размера и с произвольным отступом;
* выводит содержимое данного отображения;
* записывает в данное отображение новые данные;
* вызывает функцию выгрузки данных из памяти в файл на носителе;
* завершает свою работу;
* обрабатывает нештатные ситуации и корректно управляет ресурсами (т.е. освобождает ненужные ресурсы и т.д.).

С помощью псевдофайловой системы «/proc» найти и изучить информацию о файле отображенном в память.

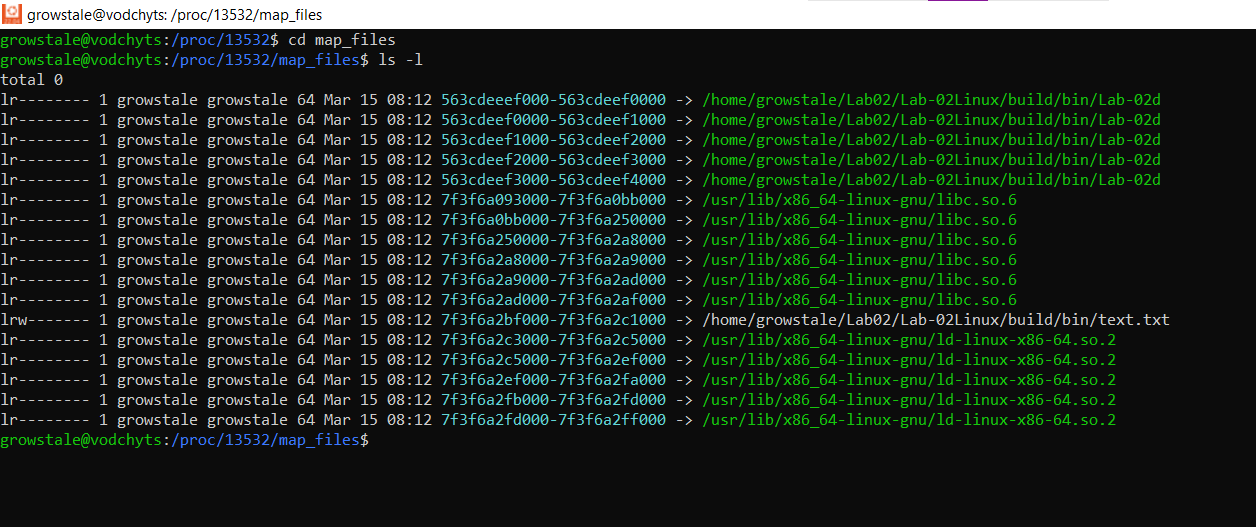
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/mman.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include <errno.h>  #define FILE\_NAME "text.txt"  #define MAP\_SIZE 8192 // 8 \* 1024  #define OFFSET 4096 // 4 \* 1024  #define NEW\_DATA "Hello, world! This is new data written to the mapped file.\n"  void handle\_error(const char \*message) {      perror(message);      exit(EXIT\_FAILURE);  }  int main() {      int fd;      struct stat file\_info;      fd = open(FILE\_NAME, O\_RDWR | O\_CREAT, 0666);      if (fd == -1) {          handle\_error("Failed to open/create file");      }      void \*mapped\_memory = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE, fd, OFFSET);      if (mapped\_memory == MAP\_FAILED) {          handle\_error("Failed to create memory mapping");      }      printf("Original content of the mapped region:\n%s\n", (char\*)mapped\_memory);      printf("Writing new data to the mapped region...\n");      memcpy(mapped\_memory, NEW\_DATA, strlen(NEW\_DATA));      printf("Process PID: %d\n", getpid());      printf("Press ENTER to continue and unmap the memory...\n");      getchar();      printf("Syncing memory to disk...\n");      if (msync(mapped\_memory, MAP\_SIZE, MS\_SYNC) == -1) {          handle\_error("Failed to sync memory to disk");      }      printf("Cleaning up...\n");      if (munmap(mapped\_memory, MAP\_SIZE) == -1) {          handle\_error("Failed to unmap memory");      }      if (close(fd) == -1) {          handle\_error("Failed to close file");      }      printf("Done. Check the file '%s'\n", FILE\_NAME);      return EXIT\_SUCCESS;  } |





Этот файл предоставляет подробную информацию об отображениях памяти (memory mappings) процесса. Каждая строка в /proc/<PID>/maps описывает отдельный регион виртуальной памяти, используемый процессом.

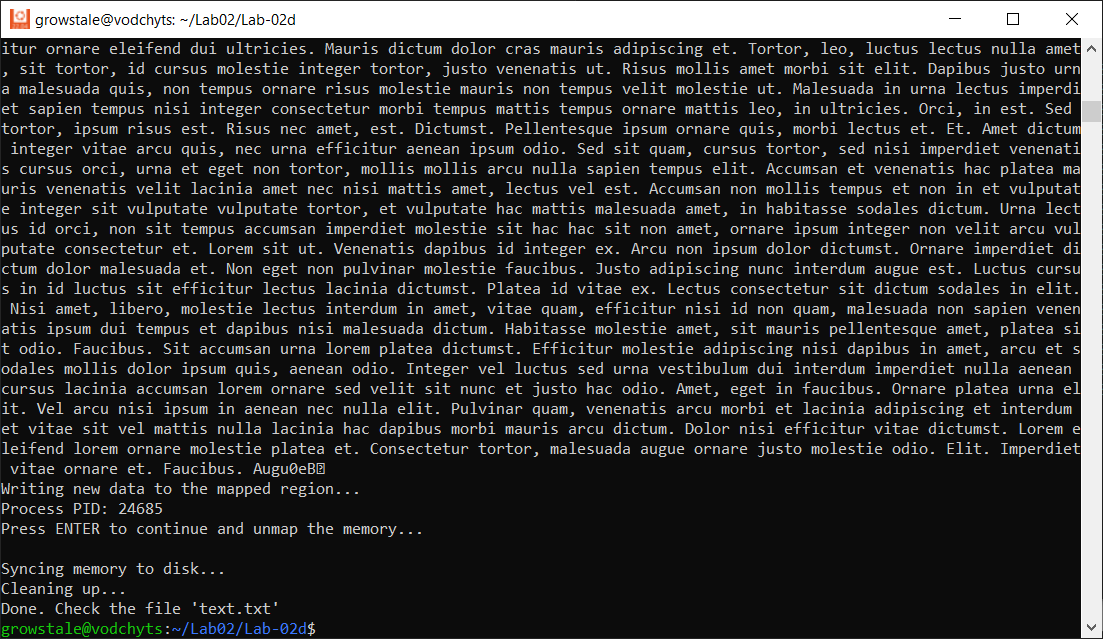
* **address**: Диапазон адресов виртуальной памяти, занимаемый регионом (например, 7f8b1c000000-7f8b1c021000).
* **perms**: Разрешения доступа к региону памяти. Это комбинация следующих флагов: read, write, execute, private, shared.
* **offset**: Смещение в файле, с которого начинается отображение. Если регион не связан с файлом (например, стек или куча), то offset обычно равен 0.
* **dev**: Старший и младший номера устройства, на котором находится файл, если регион отображает файл.
* **inode**: Номер inode файла (если регион отображает файл). Если регион не связан с файлом, то обычно это 0.
* **pathname**: Путь к файлу, который отображается в этот регион памяти.

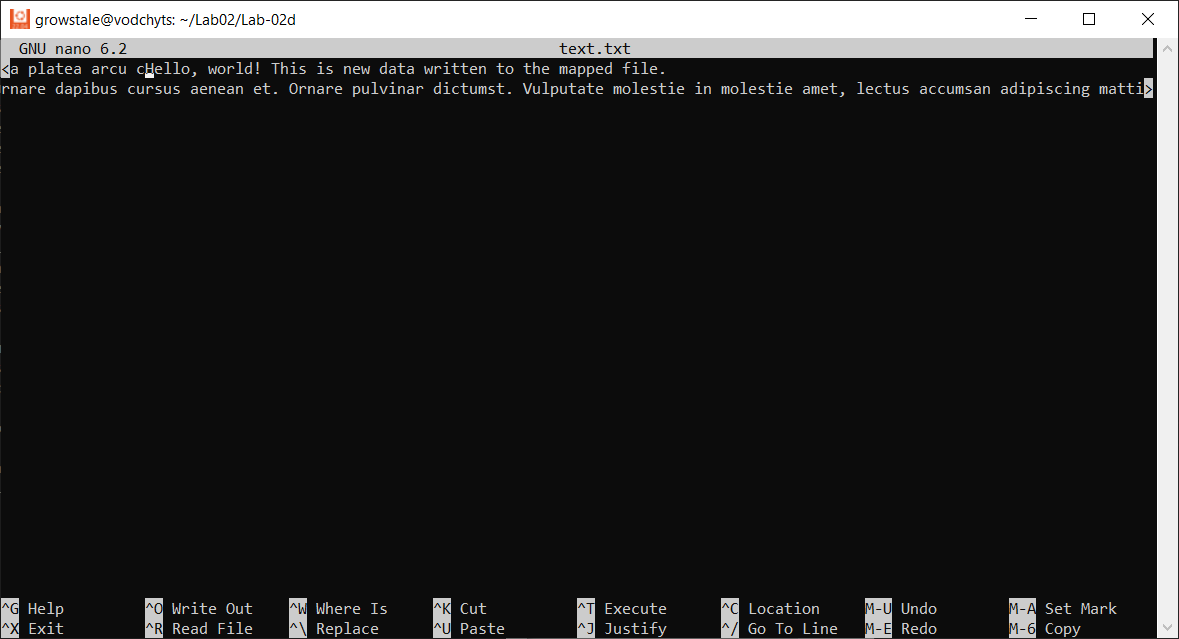


/proc/<PID>/map\_files содержит символические ссылки, имена которых представляют собой диапазоны адресов (из /proc/<PID>/maps), а сами ссылки указывают на реальные файлы на диске, которые были отображены в эти диапазоны.

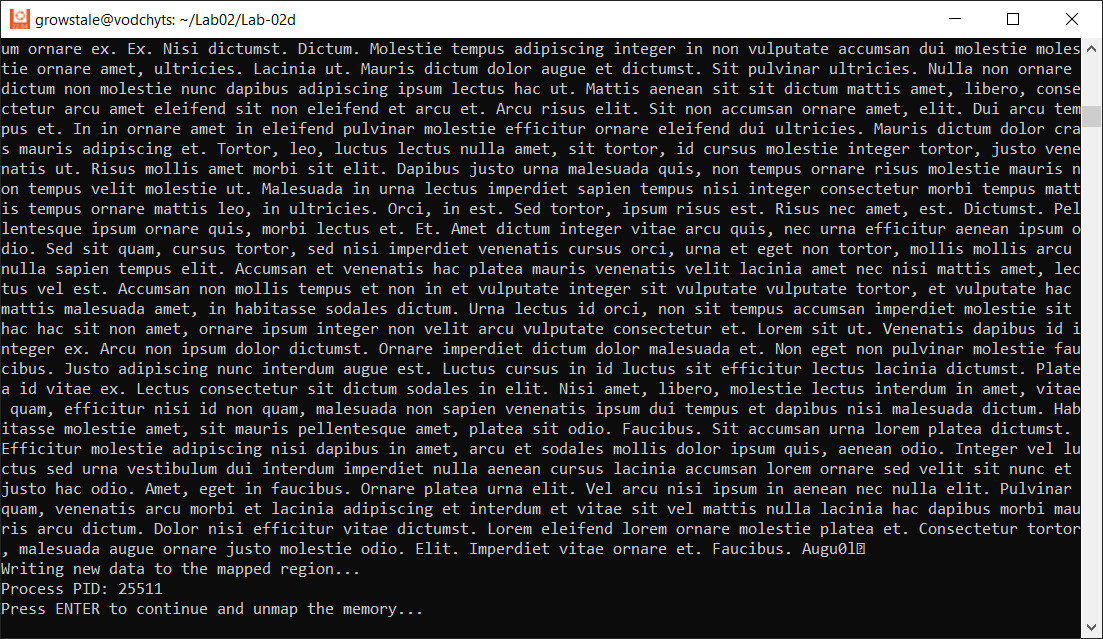
Изучить файл который был отображён в приложении.

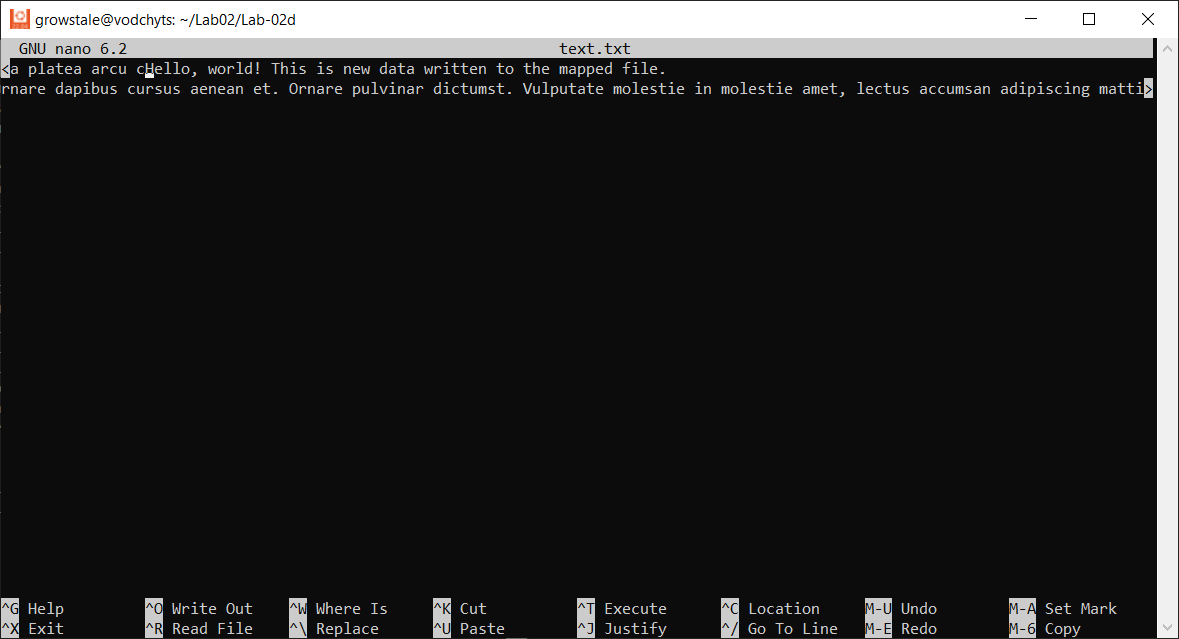
***Вопрос:*** *Были ли сохранены новые данные которые были записаны в приложении?*

**

**

Заменить приватное отображение на разделяемое и проверить изменился ли результат записи в файл.



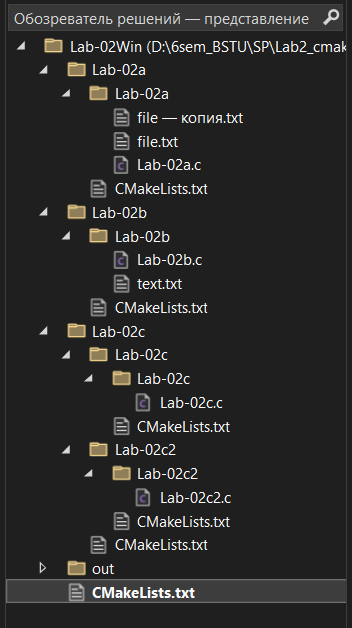


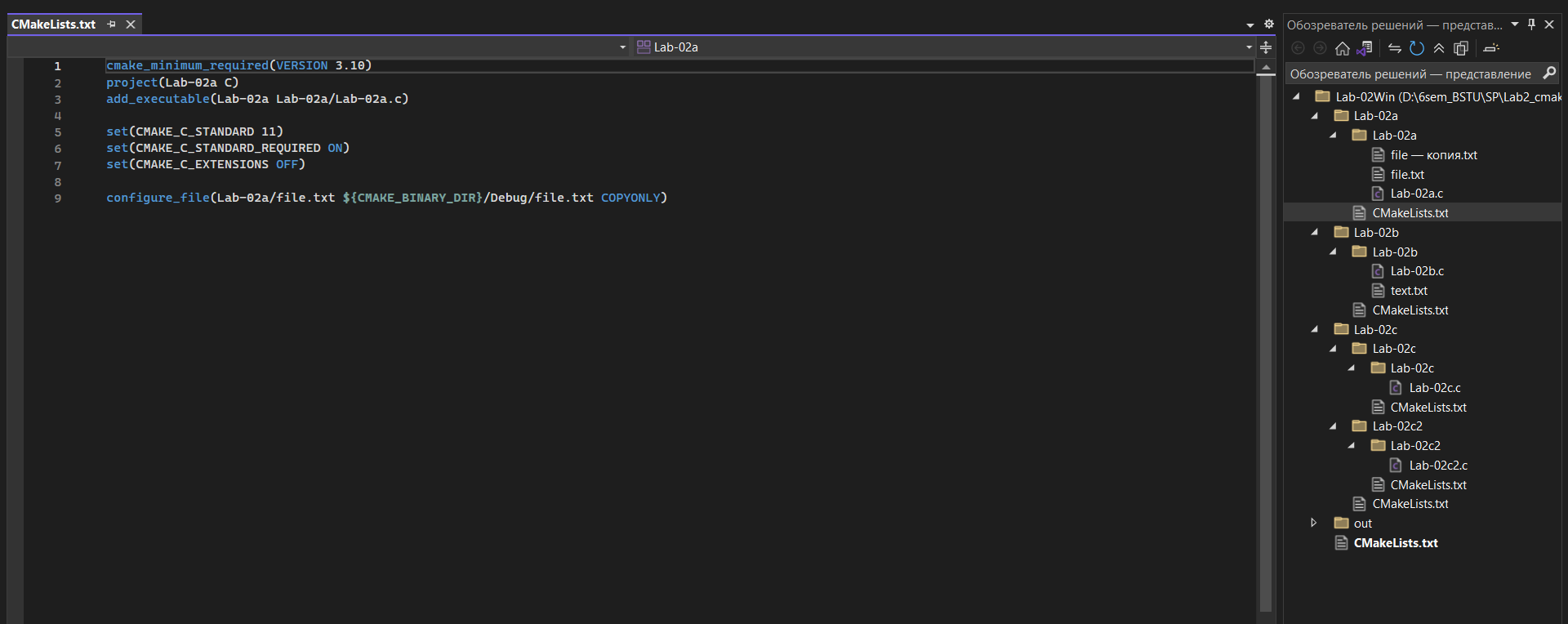
**Проект Lab-02e:**

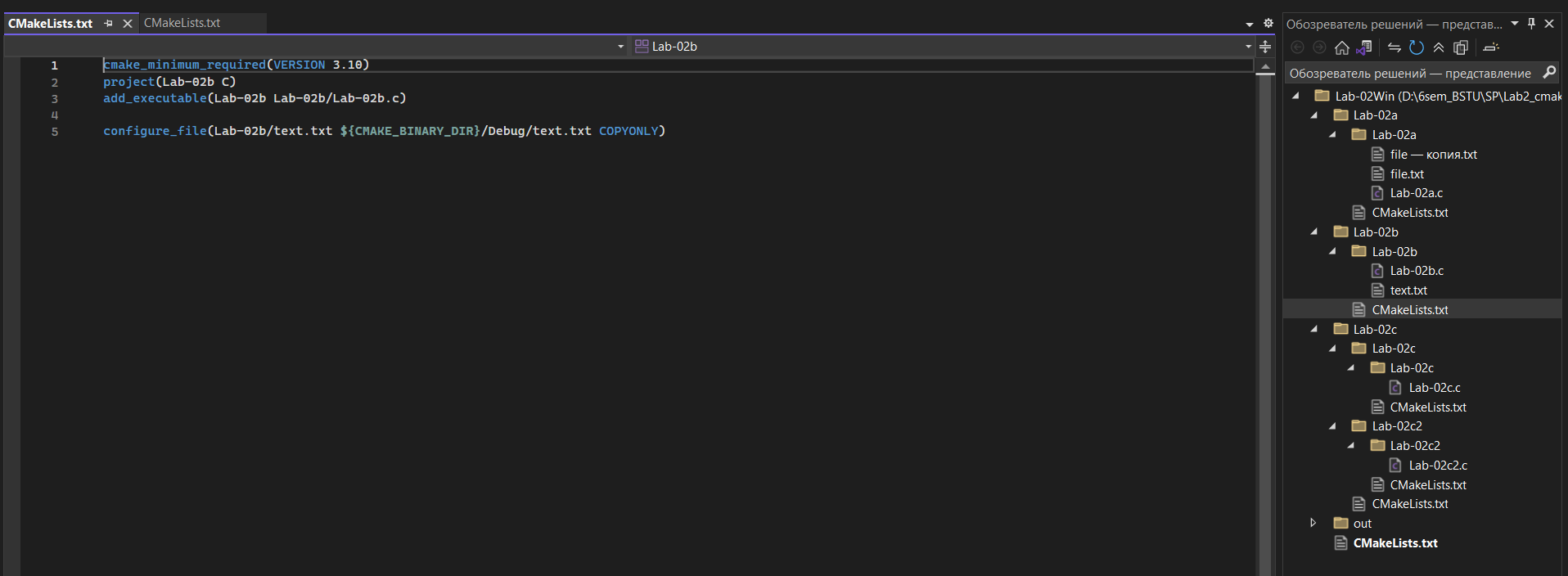
Разработать проект который состоит из двух приложений на языке C и функционально является аналогом приложения Lab-02c. Синхронизация должна быть реализована на «Семафорах».

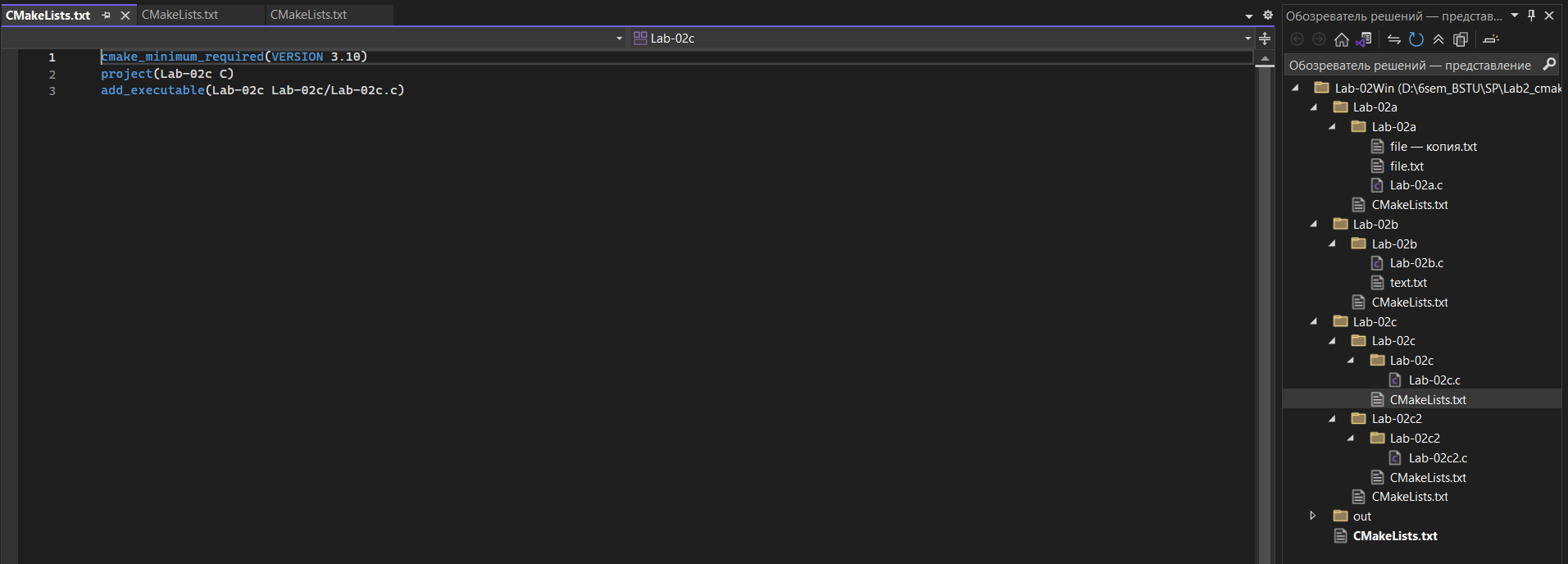
При использовании CMake должно быть выполнено следующее:

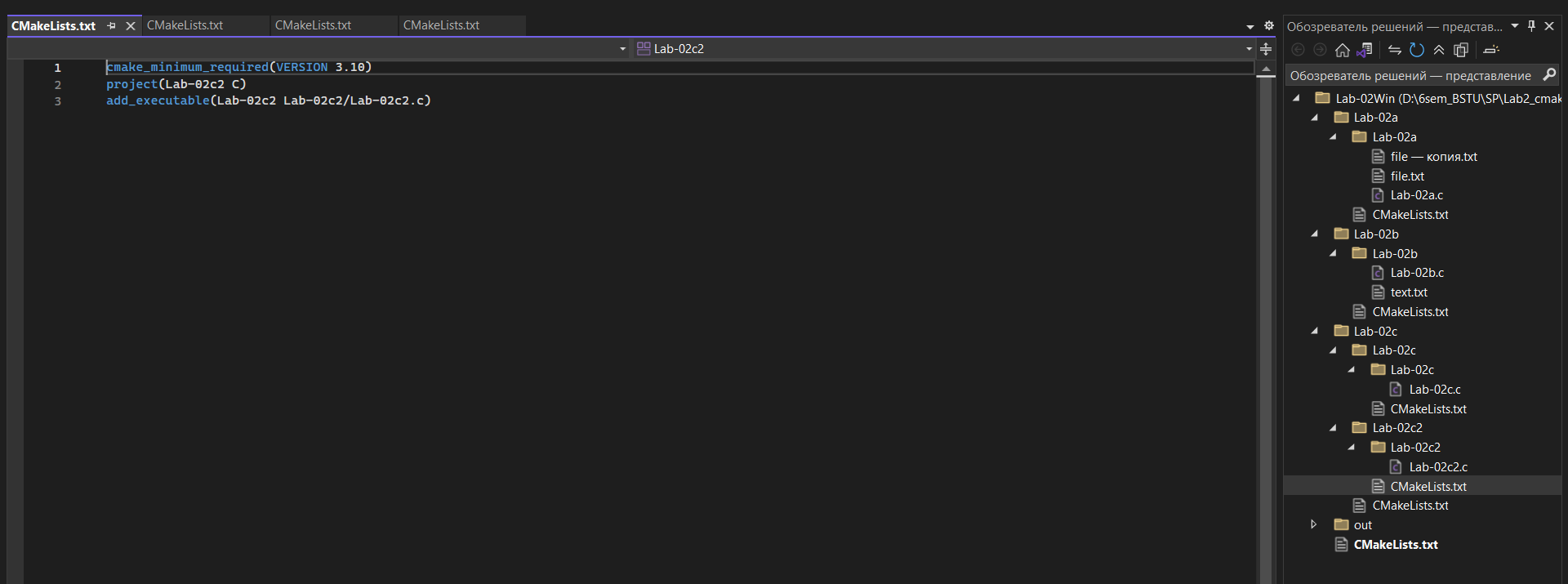
* Сборка должна быть «out-of-source»;
* Для Windows использовать генератор «Visual Studio» (для тех, у кого нет VS проще использовать генератор «[Ninja](https://ninja-build.org/)»), для Linux – «Unix Makefiles», для macOS – «XCode»;
* При сборке проектов должен использоваться clang;
* Все проекты в рамках одной ОС (например Lab-02a, Lab-02b и Lab-02c) должны быть собраны как части одного большого проекта.

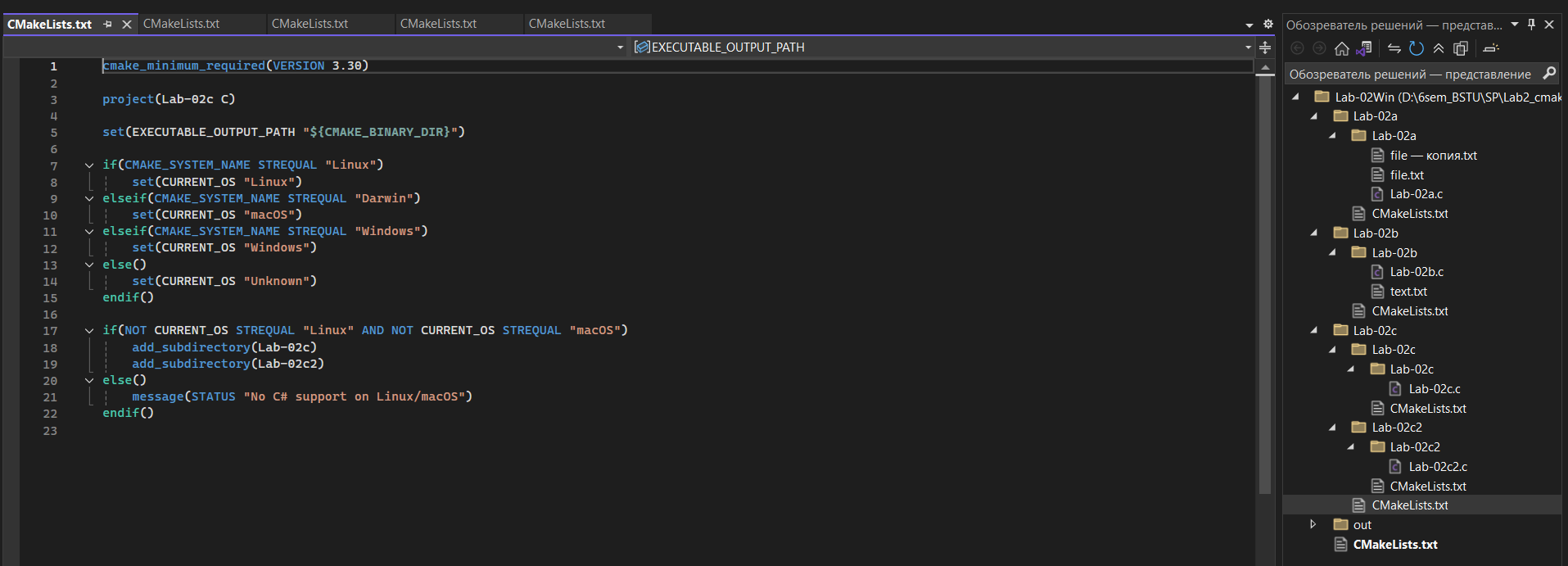
****

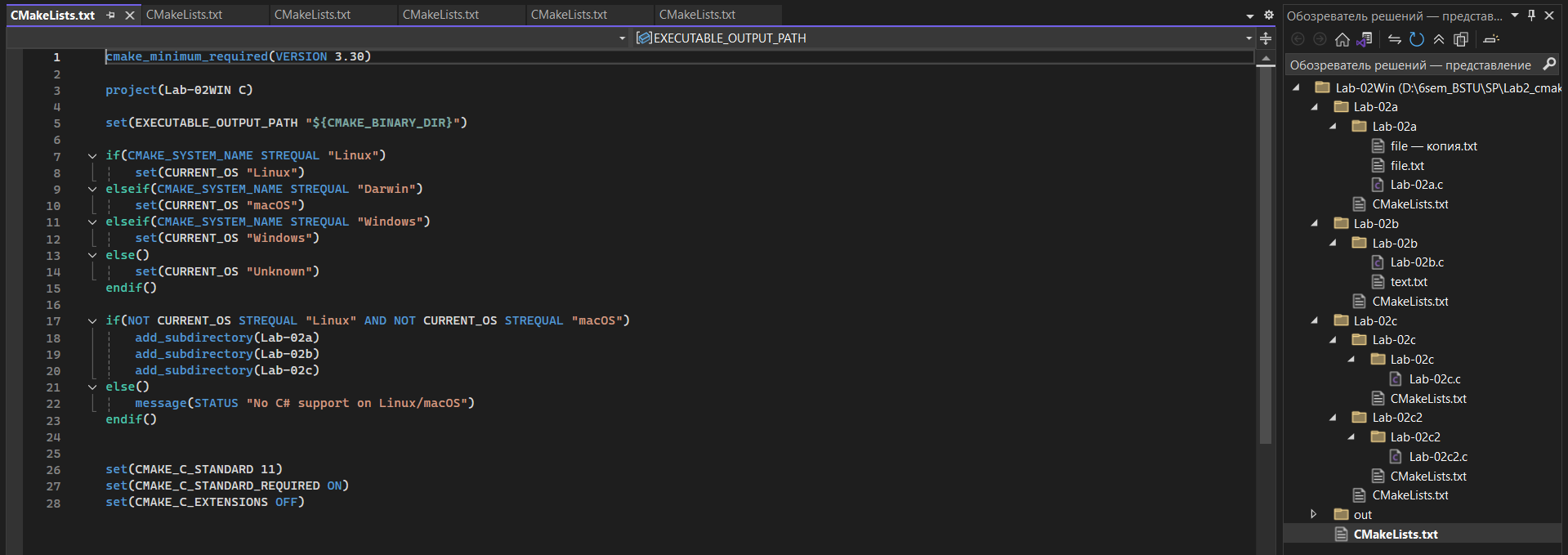
****

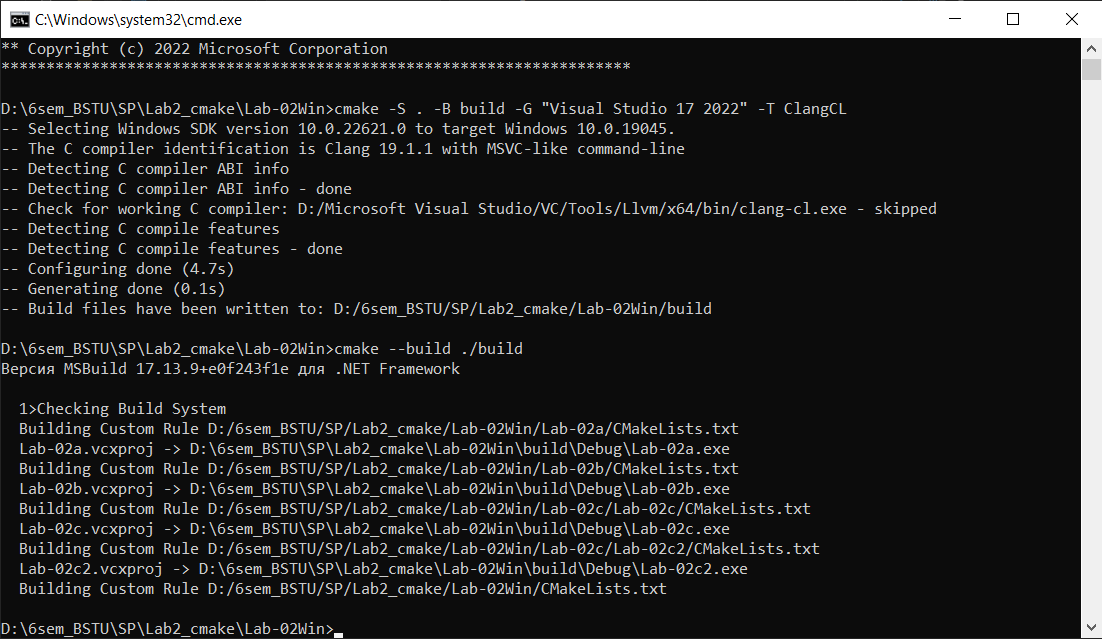
****

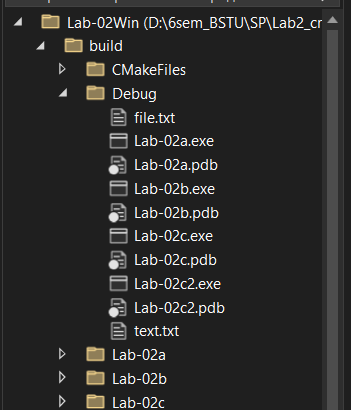
****

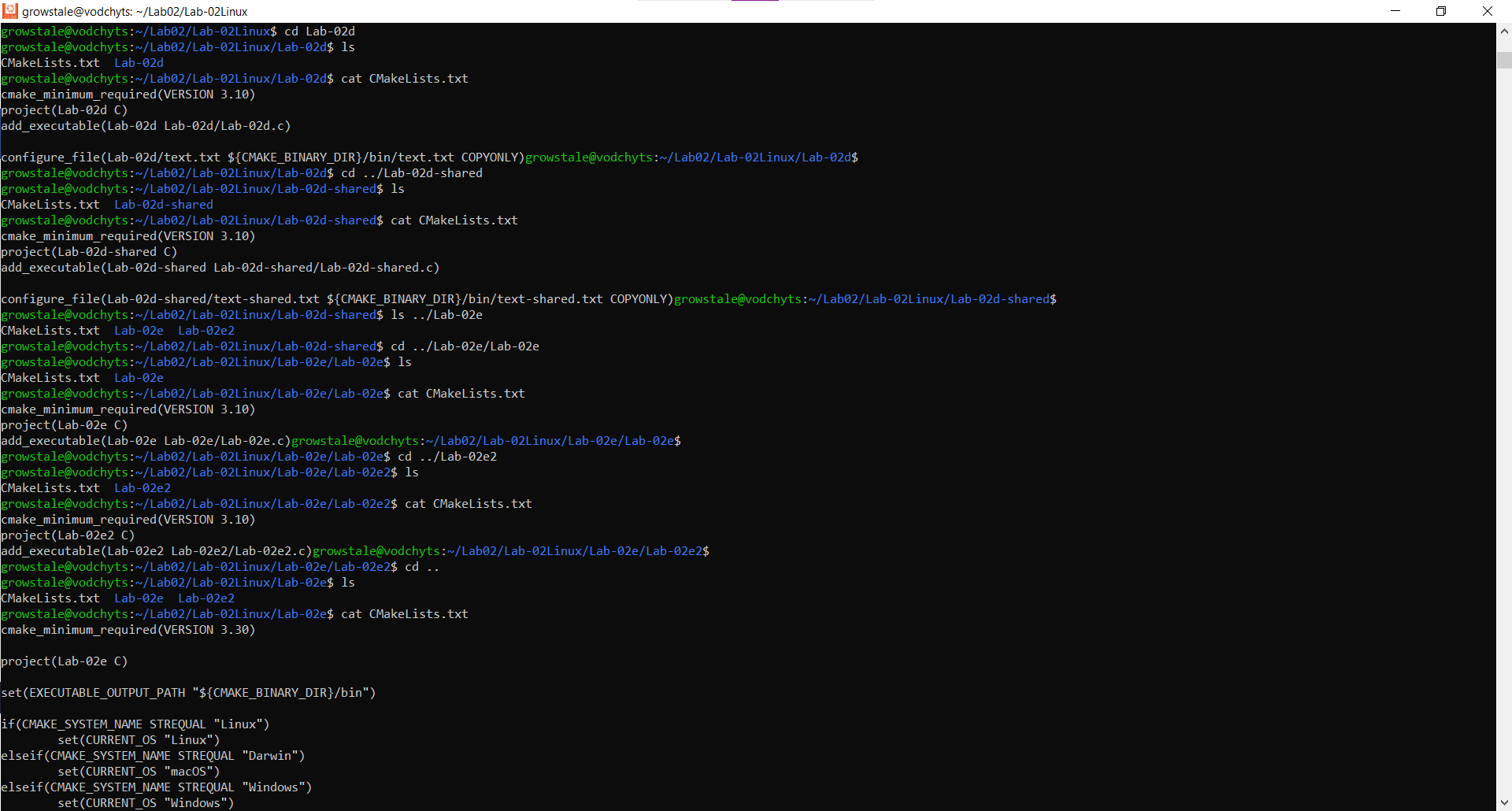
****

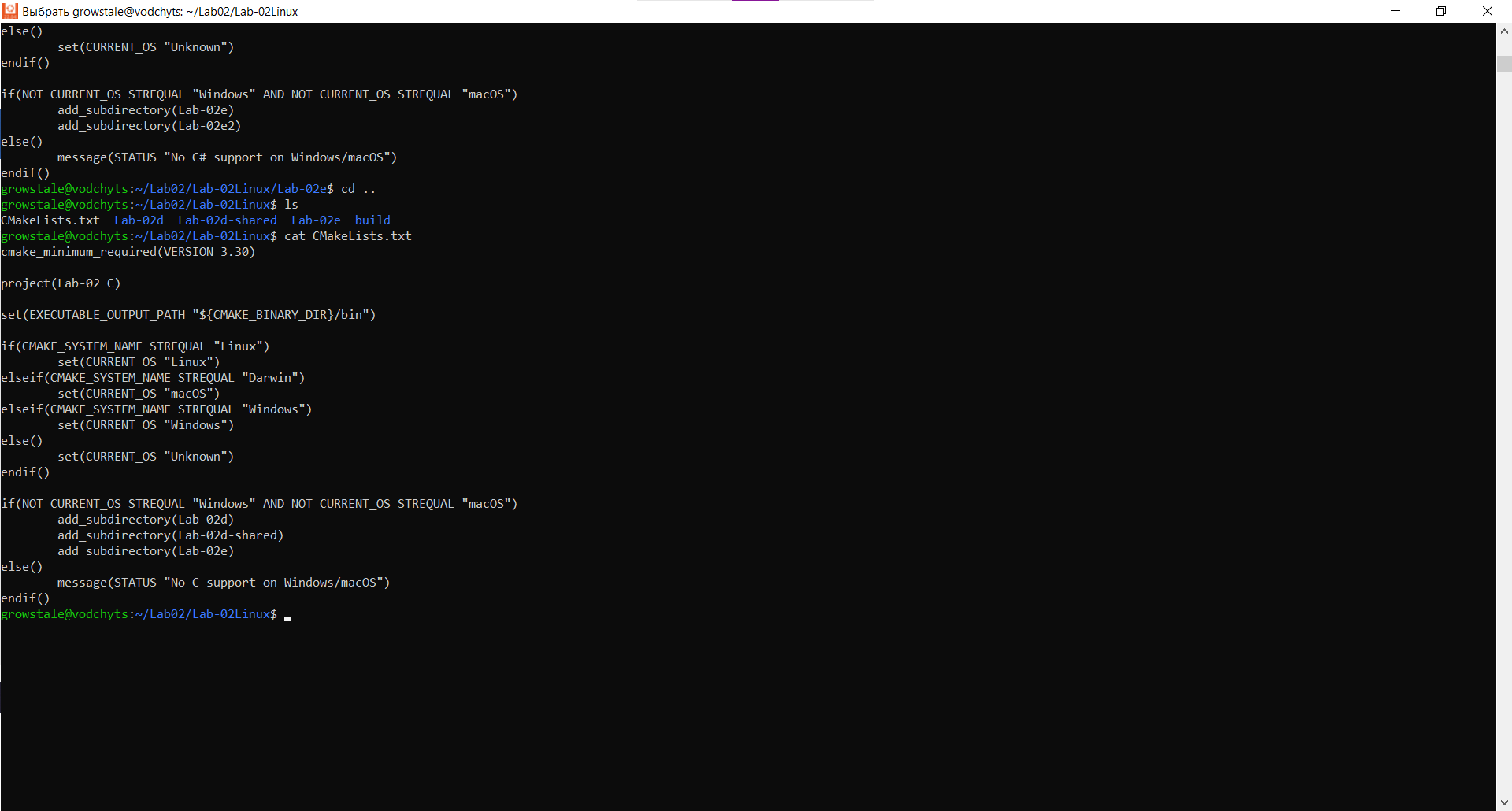
****

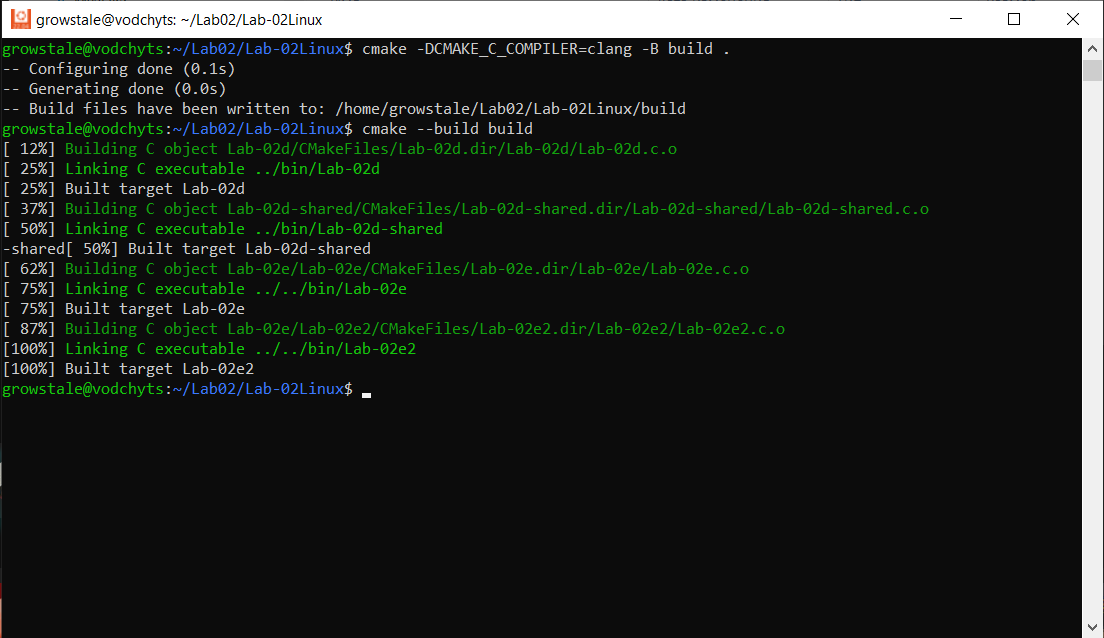
****

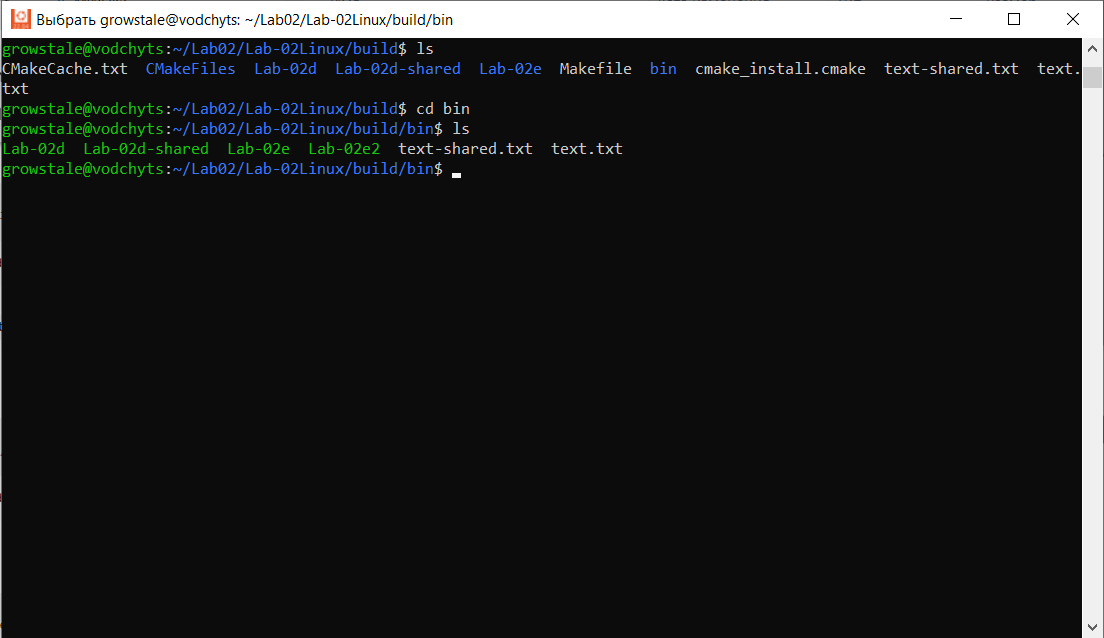
****

****

****

****

****

****

**Вопросы для контроля:**

1. Что такое файл отображенный в память?
2. Windows: что такое объект «проекции файла»?
3. Windows: что такое представление отображения?
4. Windows: какие бывают отображения?
5. Windows: какие функции входят в API для управления файлами отображенными в память?
6. Linux: какие бывают отображения и в чём их особенности?
7. Linux: какие функции входят в API для управления файлами отображенными в память?
8. Что такое Copy On Write?