Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Системное программирование

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

Минск 2024

**Лабораторная работа №2**

Знакомство с файлами отображенными в память

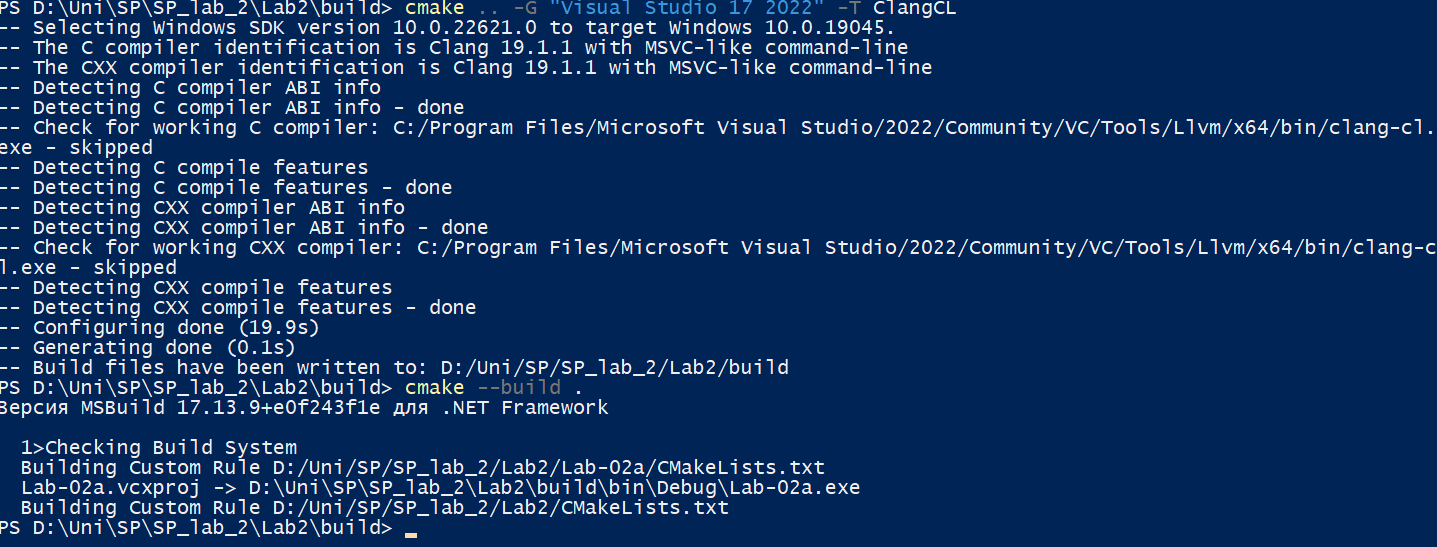
**Цель работы:** Получение практических навыков в работе с файлами отображенными в память.

**Постановка задачи для Windows:**

**Проект Lab-02a:** Разработать проект который состоит из одного приложения на языке C которое содержит код который выполняет следующее:

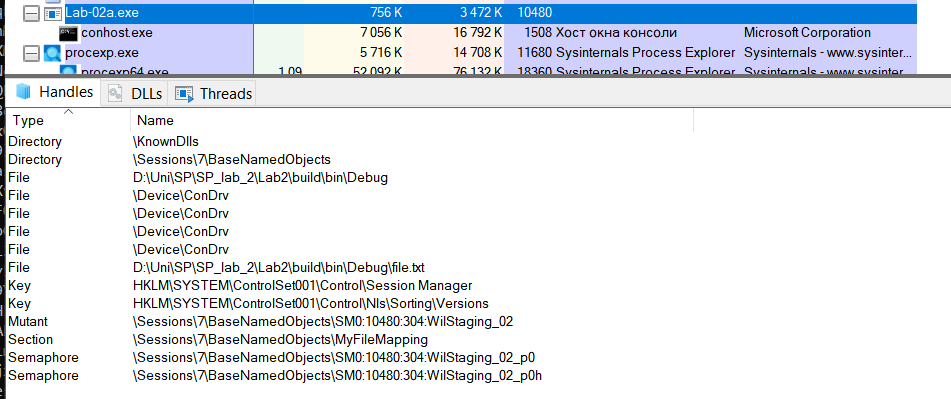
* создаёт и использует файл отображенный в память (файл должен быть текстовый);
* создаёт представление на часть файла (должен быть задан отступ и размер не равные 0 на своё усмотрение) и выводит её содержимое на консоль;
* создаёт представление на часть файла (должен быть задан отступ и размер не равные 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.).

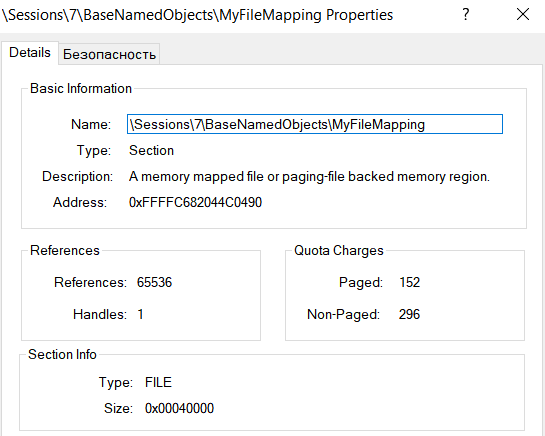
|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define VIEW\_OFFSET\_1 ((size\_t)65536) // Смещение 64 KB  #define VIEW\_SIZE\_1 ((size\_t)4096) // 4 KB для чтения  #define VIEW\_OFFSET\_2 ((size\_t)131072) // Смещение 128 KB  #define VIEW\_SIZE\_2 ((size\_t)4096) // 4 KB для записи  #define FILE\_SIZE ((size\_t)262144) // 256 KB для файла  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  void generate\_random\_text(char\* buffer, size\_t size) {  const char charset[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  for (size\_t i = 0; i < size - 1; i++) {  buffer[i] = charset[rand() % (sizeof(charset) - 1)];  }  buffer[size - 1] = '\0';  }  int main() {  HANDLE hFile = INVALID\_HANDLE\_VALUE;  HANDLE hMap = NULL;  LPVOID pView = NULL;  char randomText[FILE\_SIZE];  srand((unsigned int)time(NULL));  hFile = CreateFileW(L"file.txt", GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE) handle\_error("Failed to create file");  generate\_random\_text(randomText, FILE\_SIZE);  DWORD bytesWritten;  if (!WriteFile(hFile, randomText, FILE\_SIZE, &bytesWritten, NULL)) {  CloseHandle(hFile);  handle\_error("Failed to write random data to file");  }  hMap = CreateFileMappingW(hFile, NULL, PAGE\_READWRITE, 0, FILE\_SIZE, L"MyFileMapping");  if (hMap == NULL) {  CloseHandle(hFile);  handle\_error("Failed to create file mapping");  }  printf("File mapping created successfully\n");  pView = MapViewOfFile(hMap, FILE\_MAP\_READ, 0, VIEW\_OFFSET\_1, 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 %zu (size %zu):\n", VIEW\_OFFSET\_1, VIEW\_SIZE\_1);  for (size\_t i = 0; i < VIEW\_SIZE\_1; i++) {  putchar(((char\*)pView)[i]);  }  printf("\n");  printf("Press enter to continue... ");  getchar();  UnmapViewOfFile(pView);  pView = MapViewOfFile(hMap, FILE\_MAP\_WRITE, 0, VIEW\_OFFSET\_2, 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;  } |

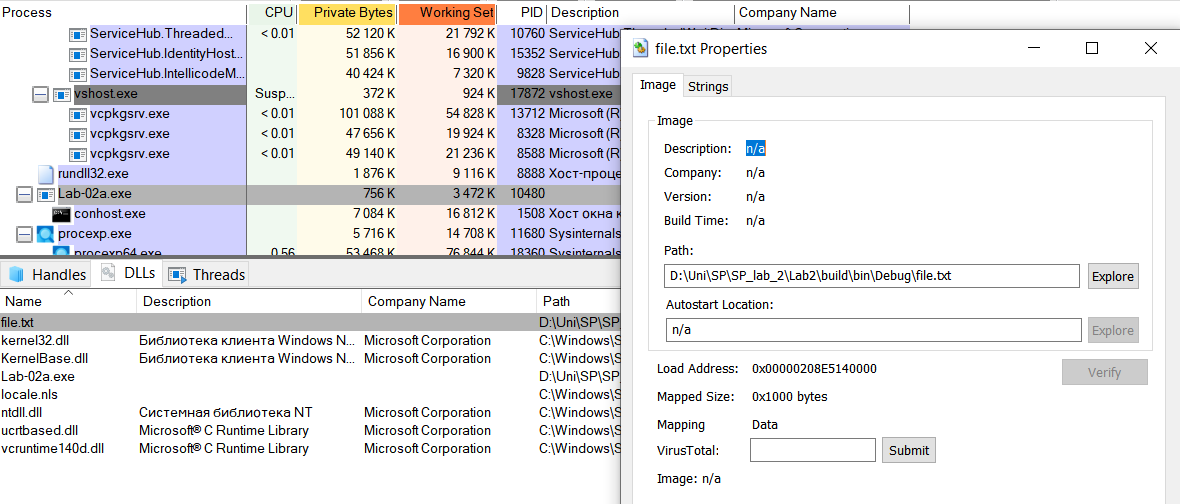


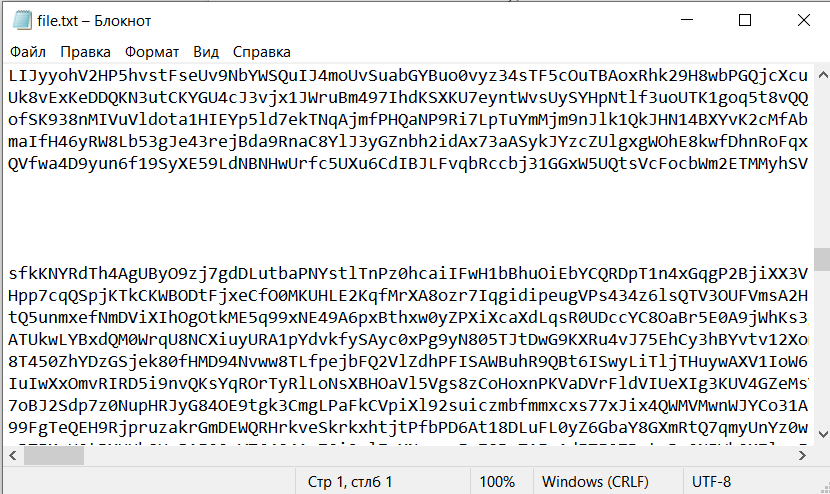
С помощью утилиты **Process Explorer** найти и изучить информацию о файле отображенном в память.

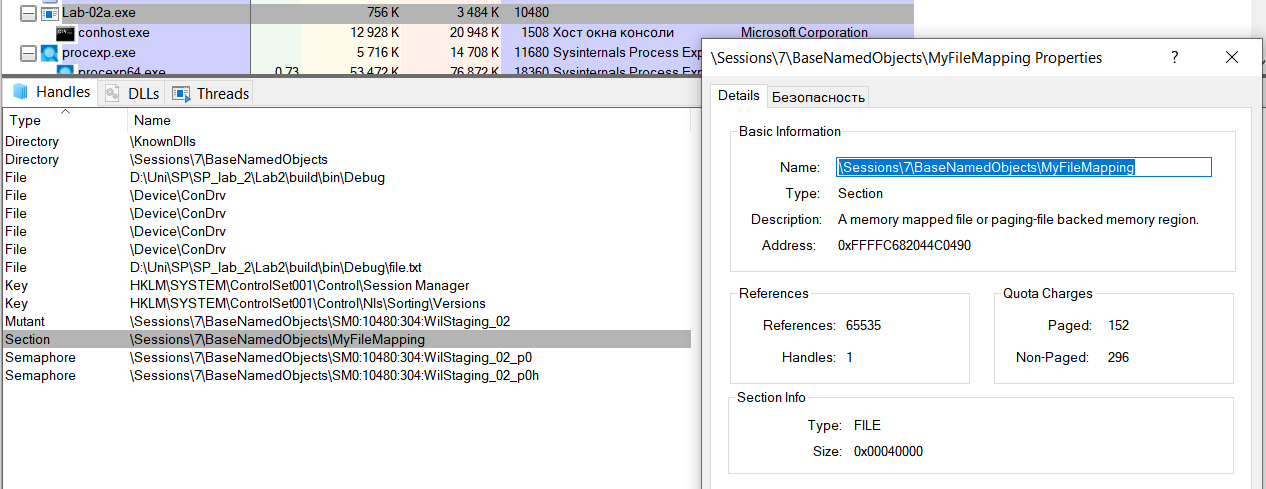


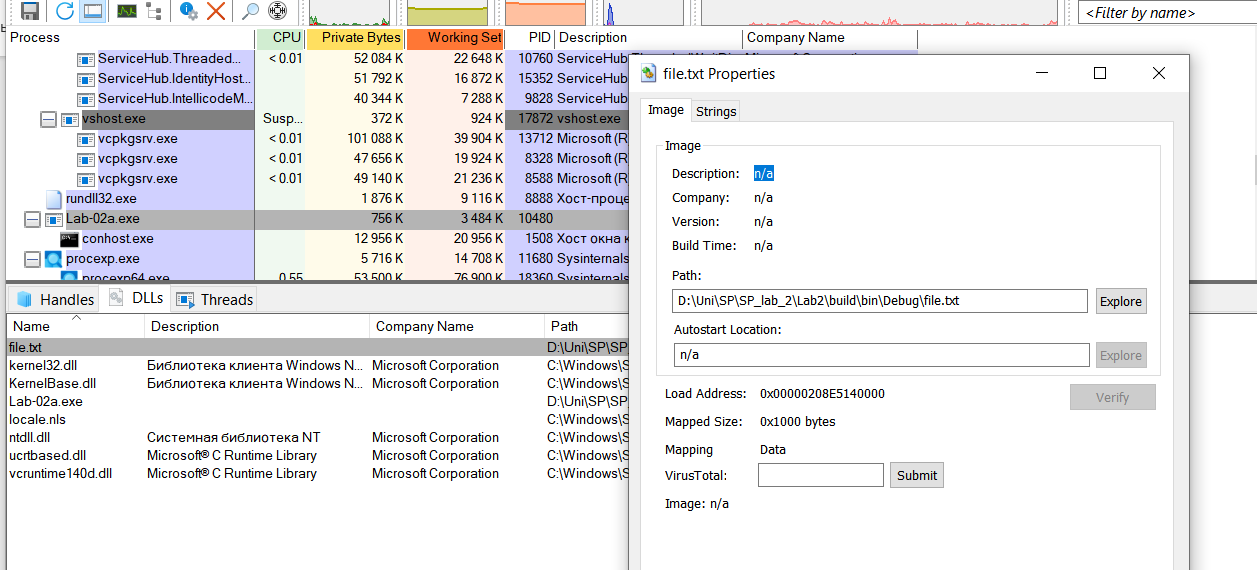


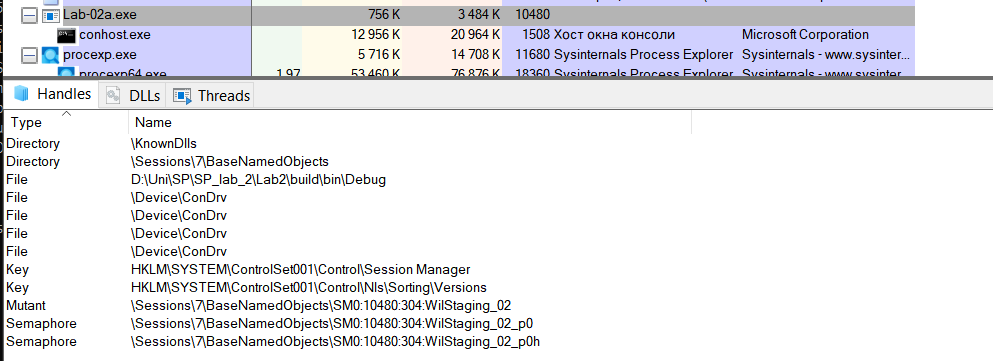


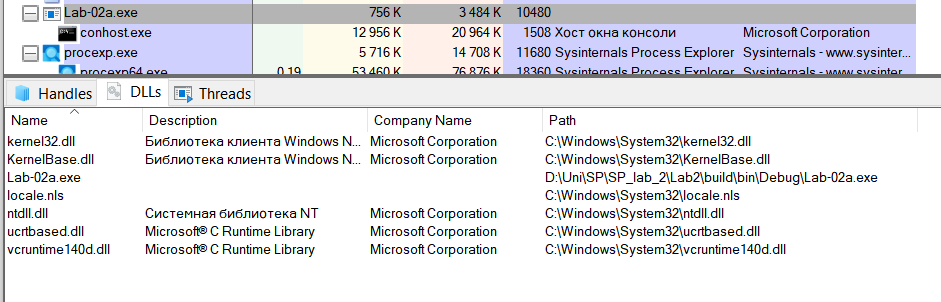












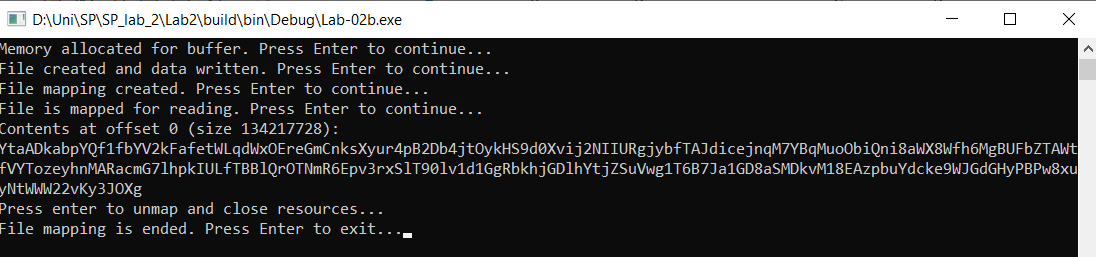
**Проект Lab-02b:** Разработать проект который состоит из одного приложения на языке C которое содержит код который выполняет следующее:

* создаёт и использует файл отображенный в память (отображаемый файл должен быть текстовый и размером не менее 512 МиБ). 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.

|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define FILE\_SIZE ((size\_t)536870912) // 512 MiB для файла  #define VIEW\_OFFSET ((size\_t)0) // Смещение  #define VIEW\_SIZE ((size\_t)134217728) // 128 MiB для части файла  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  void generate\_random\_text(char\* buffer, size\_t size) {  const char charset[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  for (size\_t i = 0; i < size; i++) {  buffer[i] = charset[rand() % (sizeof(charset) - 1)];  }  }  int main() {  HANDLE hFile = INVALID\_HANDLE\_VALUE;  HANDLE hMap = NULL;  LPVOID pView = NULL;  char\* buffer = (char\*)malloc(FILE\_SIZE);  if (!buffer) handle\_error("Memory allocation failed");  srand((unsigned int)time(NULL));  generate\_random\_text(buffer, FILE\_SIZE);  hFile = CreateFileW(L"fileb.txt", GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE) handle\_error("Failed to create file");  DWORD bytesWritten;  if (!WriteFile(hFile, buffer, FILE\_SIZE, &bytesWritten, NULL)) {  CloseHandle(hFile);  free(buffer);  handle\_error("Failed to write random data to file");  }  free(buffer);  hMap = CreateFileMappingW(hFile, NULL, PAGE\_READWRITE, 0, FILE\_SIZE, L"Lab2bFileMapping");  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 %zu (size %zu):\n", VIEW\_OFFSET, VIEW\_SIZE);  for (size\_t i = 0; i < 256; i++) {  putchar(((char\*)pView)[i]);  }  printf("\n");  printf("Press enter to continue... ");  getchar();  UnmapViewOfFile(pView);  CloseHandle(hMap);  CloseHandle(hFile);  printf("File mapping is ended\n");  printf("Press enter to continue... ");  getchar();  return 0;  } |

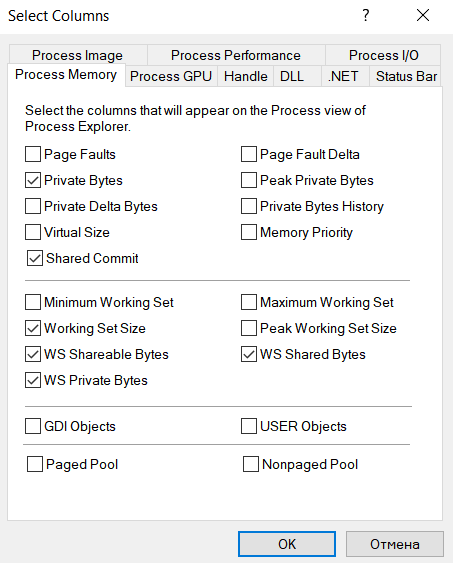
Вторая версия:

|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define FILE\_SIZE ((size\_t)536870912) // 512 MiB  #define VIEW\_OFFSET ((size\_t)0) // Смещение  #define VIEW\_SIZE ((size\_t)134217728) // 128 MiB  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  void generate\_random\_text(char\* buffer, size\_t size) {  const char charset[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  for (size\_t i = 0; i < size; i++) {  buffer[i] = charset[rand() % (sizeof(charset) - 1)];  }  }  int main() {  HANDLE hFile = INVALID\_HANDLE\_VALUE;  HANDLE hMap = NULL;  LPVOID pView = NULL;  char\* buffer = (char\*)malloc(FILE\_SIZE);  if (!buffer) handle\_error("Memory allocation failed");  srand((unsigned int)time(NULL));  generate\_random\_text(buffer, FILE\_SIZE);  hFile = CreateFileW(L"fileb.txt", GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE) {  free(buffer);  handle\_error("Failed to create file");  }  DWORD bytesWritten;  if (!WriteFile(hFile, buffer, FILE\_SIZE, &bytesWritten, NULL)) {  CloseHandle(hFile);  free(buffer);  handle\_error("Failed to write random data to file");  }  free(buffer);  printf("\nFile created. Press enter to continue... ");  getchar();  hMap = CreateFileMappingW(hFile, NULL, PAGE\_READWRITE, 0, FILE\_SIZE, L"Lab2bFileMapping");  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 %zu (size %zu):\n", VIEW\_OFFSET, VIEW\_SIZE);  fwrite(pView, 1, VIEW\_SIZE, stdout);  printf("\nPress enter to continue... ");  getchar();  UnmapViewOfFile(pView);  CloseHandle(hMap);  CloseHandle(hFile);  printf("File mapping is ended\n");  printf("Press enter to exit... ");  getchar();  return 0;  } |

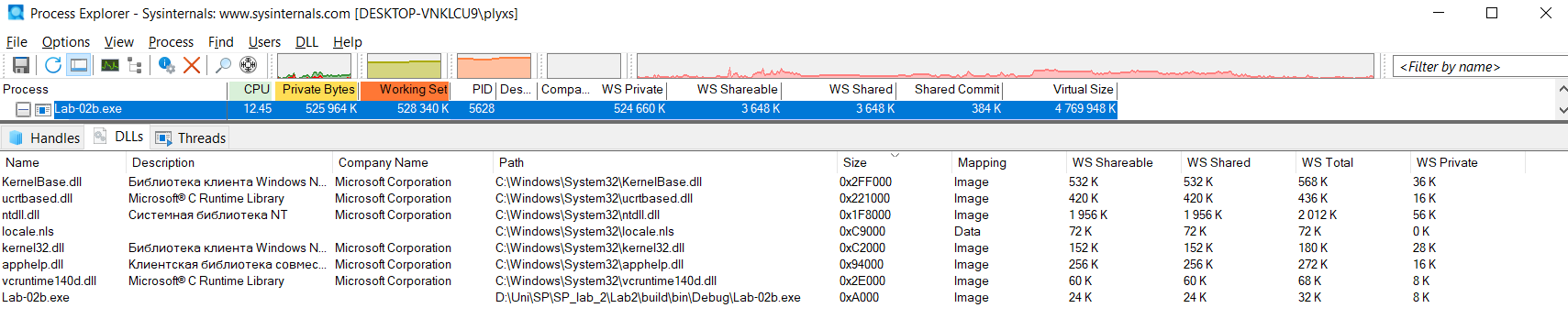


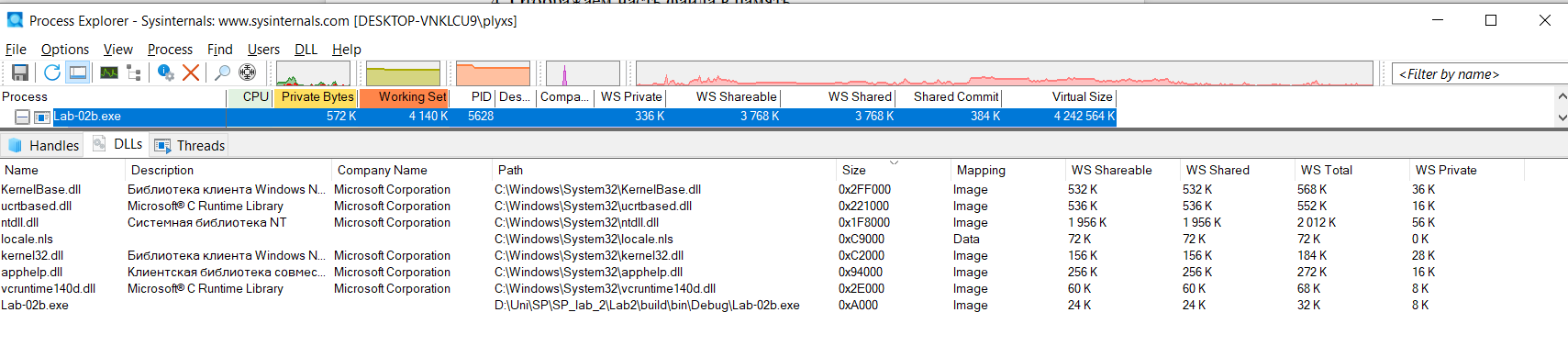
С помощью утилиты Process Explorer найти и изучить информацию о том, в какой момент времени процесс начинает потреблять больше всего физической памяти. (Например, сделайте паузы между этапами с помощью getchar)

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-).

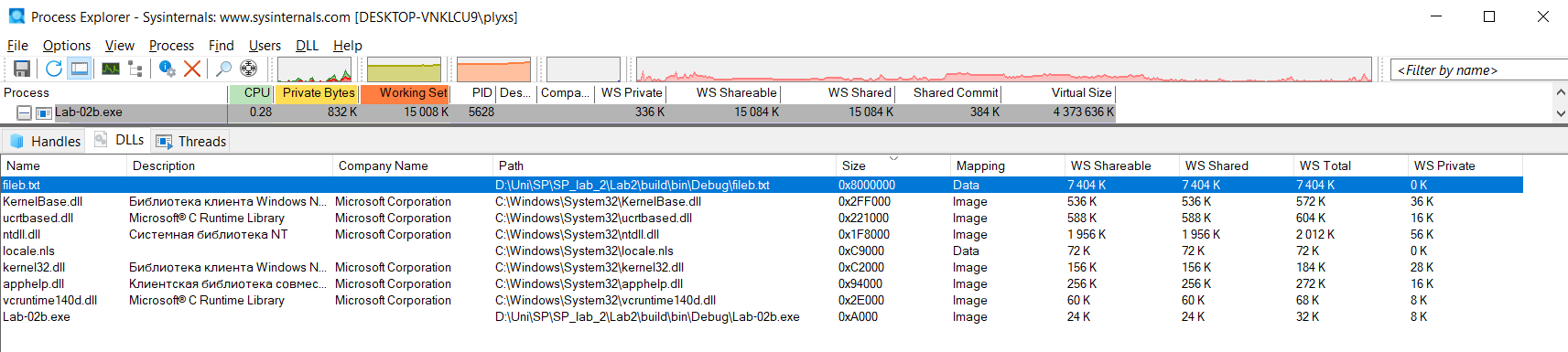


512 MiB в Private Working Set

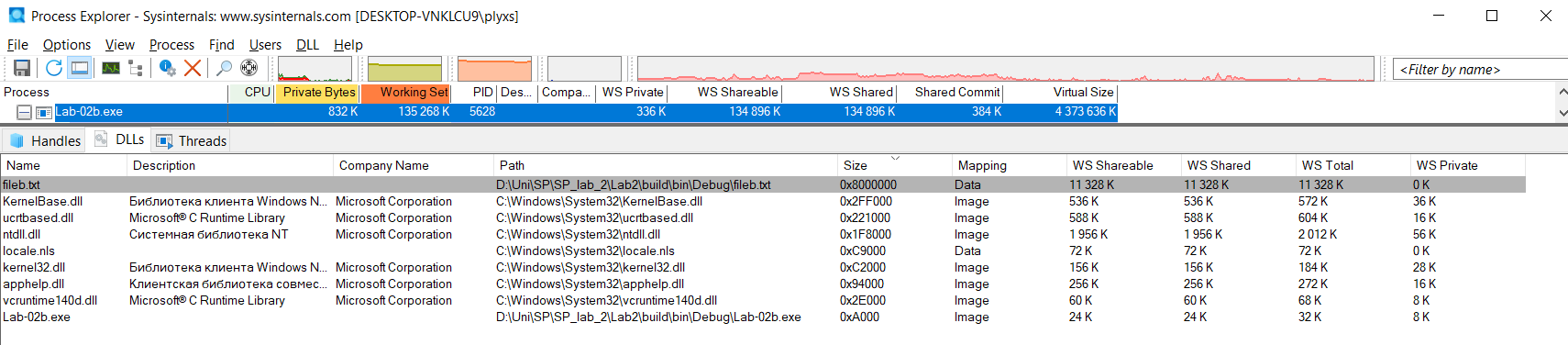


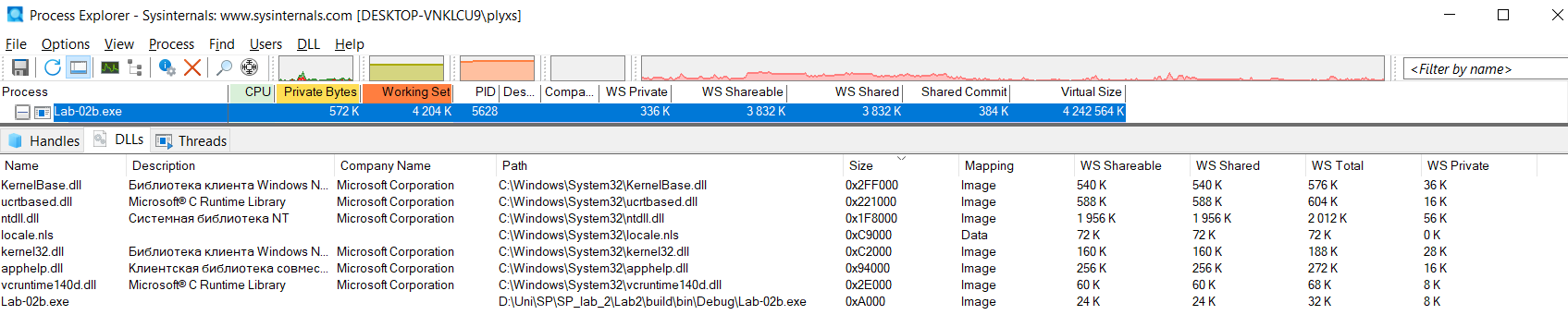
ПослеWriteFile

Working Set растет по мере чтения



Shared WS += 128 MiB (view)

Все показатели возвращаются к исходным



**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** – 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.

**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.

Низкий объём **Private Bytes** и **WS Private** означает, что процесс почти не использует свою личную память.

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.

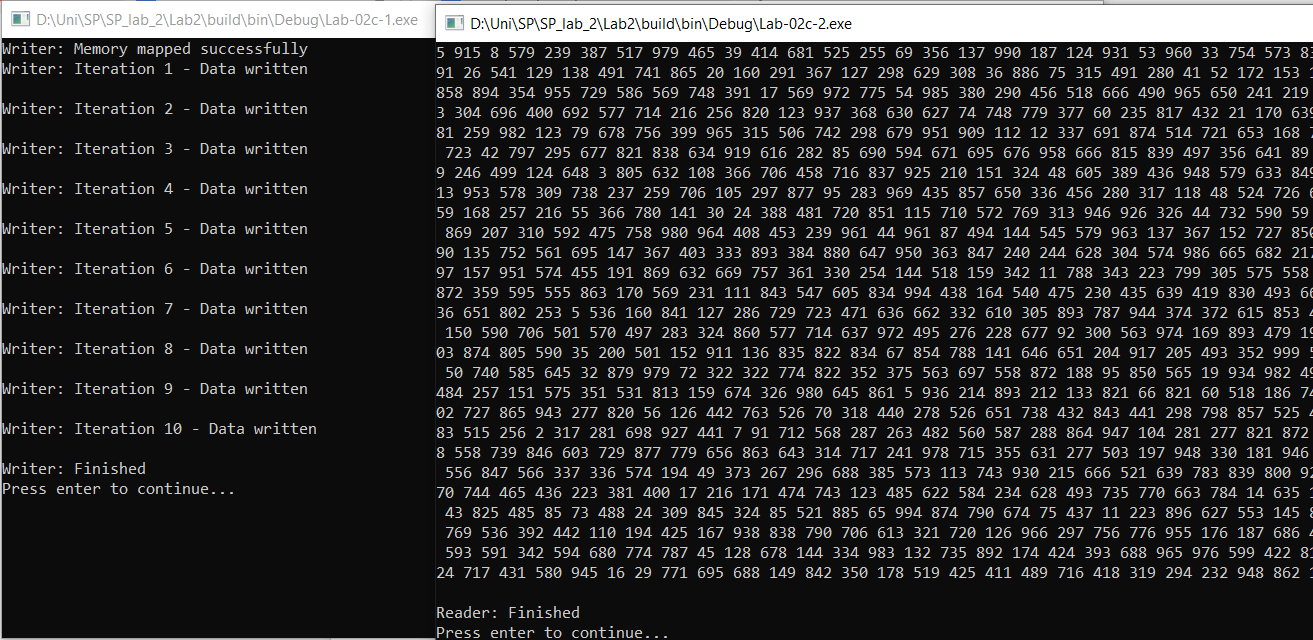
|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #define FILE\_MAPPING\_NAME L"Lab-02"  #define MUTEX\_NAME L"Lab-02-Mutex"  #define VIEW\_SIZE 65536 // 64 КиБ  #define ITERATIONS 10 // 10 итераций  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  void start\_reader\_process() {  STARTUPINFOW si = { sizeof(si) };  PROCESS\_INFORMATION pi;  if (!CreateProcessW(L"Lab-02c-2.exe", NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi)) {  handle\_error("Failed to start Lab-02c-2.exe");  }  CloseHandle(pi.hProcess);  CloseHandle(pi.hThread);  printf("Writer: Reader process started successfully\n");  }  int main() {  HANDLE hMap = NULL, hMutex = NULL;  LPVOID pView = NULL;  hMutex = CreateMutexW(NULL, FALSE, MUTEX\_NAME);  if (hMutex == NULL) handle\_error("Failed to create mutex");  hMap = CreateFileMappingW(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, VIEW\_SIZE, FILE\_MAPPING\_NAME);  if (hMap == NULL) handle\_error("Failed to create file mapping");  pView = MapViewOfFile(hMap, FILE\_MAP\_WRITE, 0, 0, VIEW\_SIZE);  if (pView == NULL) handle\_error("Failed to map view");  printf("Writer: Memory mapped successfully\n");  //start\_reader\_process();  for (int i = 0; i < ITERATIONS; i++) {  WaitForSingleObject(hMutex, INFINITE);  for (size\_t j = 0; j < VIEW\_SIZE / sizeof(int); j++) {  ((int\*)pView)[j] = rand() % 1000;  }  printf("Writer: Iteration %d - Data written\n", i + 1);  ReleaseMutex(hMutex);  getchar();  }  UnmapViewOfFile(pView);  CloseHandle(hMap);  CloseHandle(hMutex);  printf("Writer: Finished\n");  printf("Press enter to continue... ");  getchar();  return 0;  } |

**Второе приложение должно:**

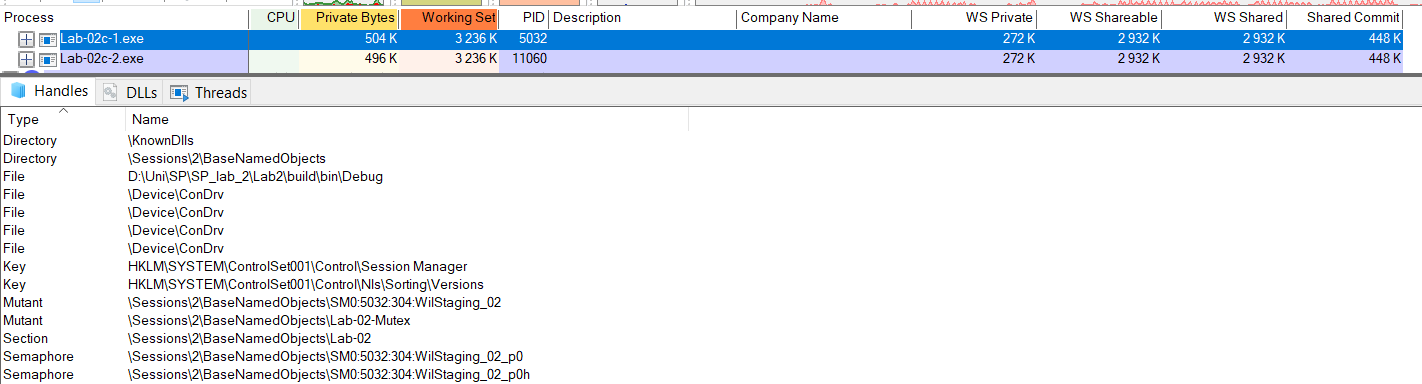
* открывать существующее отображение на базе страничного файла с именем «Lab-02»;
* создавать представление размером ровно 64 КиБ с отступом 0;
* в цикле на 10 итераций считывать массив целых чисел размером 640 КиБ (по 64 КиБ на каждой итерации) и выводить на консоль;
* освобождать все ресурсы и завершать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

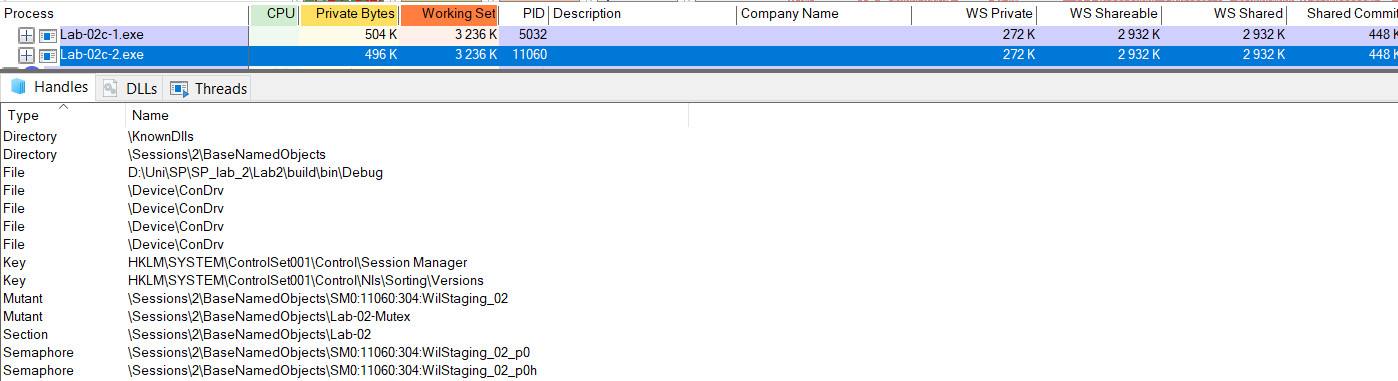
|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #define FILE\_MAPPING\_NAME L"Lab-02"  #define MUTEX\_NAME L"Lab-02-Mutex"  #define VIEW\_SIZE 65536 // 64 КиБ  #define ITERATIONS 10 // 10 итераций  void handle\_error(const char\* message) {  fprintf(stderr, "%s (Error code: %lu)\n", message, GetLastError());  exit(EXIT\_FAILURE);  }  int main() {  HANDLE hMap = NULL, hMutex = NULL;  LPVOID pView = NULL;  hMutex = OpenMutexW(SYNCHRONIZE, FALSE, MUTEX\_NAME);  if (hMutex == NULL) handle\_error("Failed to open mutex");  hMap = OpenFileMappingW(FILE\_MAP\_READ, FALSE, FILE\_MAPPING\_NAME);  if (hMap == NULL) handle\_error("Failed to open file mapping");  pView = MapViewOfFile(hMap, FILE\_MAP\_READ, 0, 0, VIEW\_SIZE);  if (pView == NULL) handle\_error("Failed to map view");  printf("Reader: Memory mapped successfully\n");  for (int i = 0; i < ITERATIONS; i++) {  WaitForSingleObject(hMutex, INFINITE);  printf("Reader: Iteration %d - Data read: ", i + 1);  for (size\_t j = 0; j < VIEW\_SIZE / sizeof(int); j++) {  printf("%d ", ((int\*)pView)[j]);  }  printf("\n");  ReleaseMutex(hMutex);  getchar();  }  UnmapViewOfFile(pView);  CloseHandle(hMap);  CloseHandle(hMutex);  printf("Reader: Finished\n");  printf("Press enter to continue... ");  getchar();  return 0;  } |

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

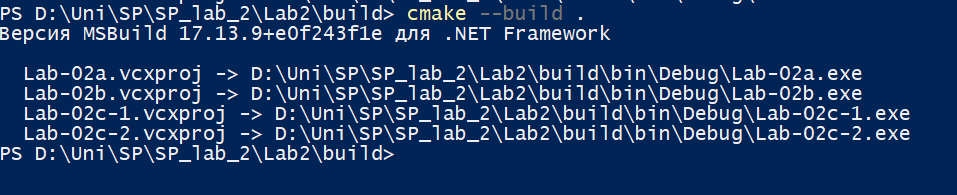


С помощью утилиты Process Explorer найти и изучить информацию об отображеннии на основе страничных файлов. (Например, сделайте паузы между этапами с помощью getchar)





Сборка:

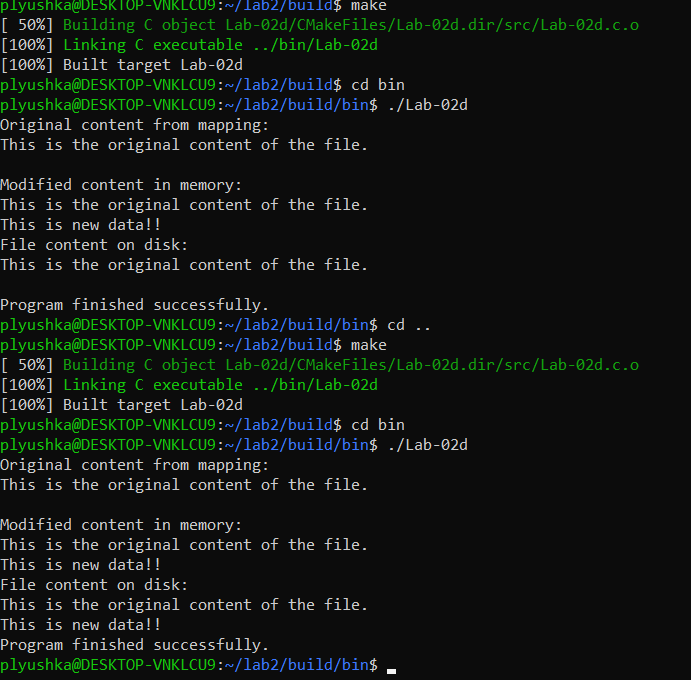


**Постановка задачи для Linux**

**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.).

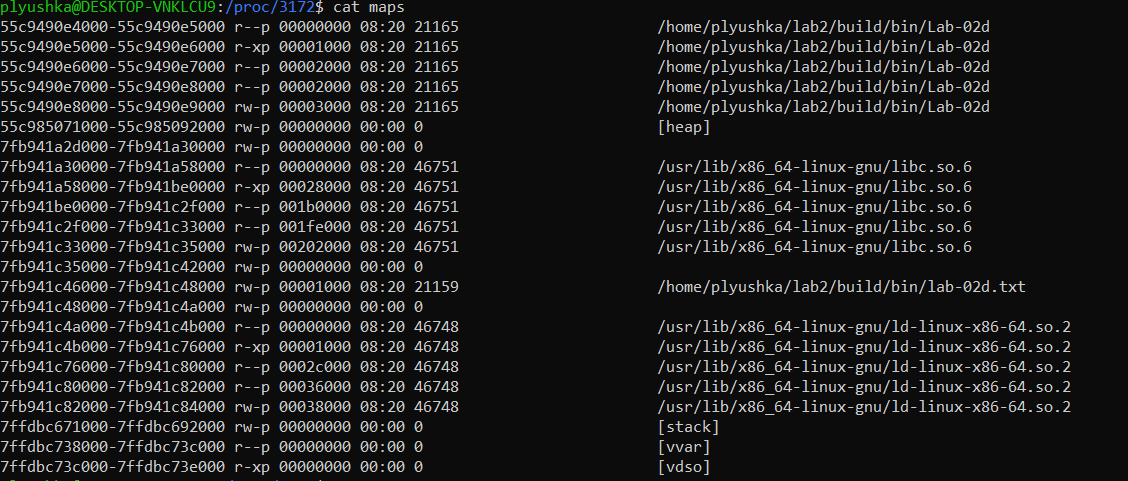
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  #include <sys/mman.h>  #include <sys/stat.h>  #include <string.h>  #include <errno.h>  #define FILENAME "lab-02d.txt"  #define FILE\_SIZE 8192  #define OFFSET 4096  void handle\_error(const char\* message) {  perror(message);  exit(EXIT\_FAILURE);  }  int main() {  int fd = open(FILENAME, O\_RDWR | O\_CREAT | O\_TRUNC, 0666);  if (fd == -1) {  handle\_error("Failed to open file");  }  if (ftruncate(fd, FILE\_SIZE) == -1) {  close(fd);  handle\_error("Failed to set file size");  }  const char\* initial\_text = "This is the original content of the file.\n";  if (pwrite(fd, initial\_text, strlen(initial\_text), OFFSET) == -1) {  close(fd);  handle\_error("Failed to write initial data to file");  }  void\* mapped = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE, fd, OFFSET);  if (mapped == MAP\_FAILED) {  close(fd);  handle\_error("Failed to map file to memory");  }  close(fd);  printf("Original content from mapping:\n%.\*s\n", (int)strlen(initial\_text), (char\*)mapped);  char\* data = (char\*)mapped;  const char\* new\_data = "This is new data!!";  strncpy(data + strlen(initial\_text), new\_data, strlen(new\_data));  printf("Modified content in memory:\n%.\*s\n", (int)(strlen(initial\_text) + strlen(new\_data)), (char\*)mapped);  if (munmap(mapped, FILE\_SIZE) == -1) {  handle\_error("Failed to unmap memory");  }  fd = open(FILENAME, O\_RDONLY);  if (fd == -1) {  handle\_error("Failed to open file for verification");  }  char file\_content[FILE\_SIZE];  ssize\_t bytes\_read = pread(fd, file\_content, FILE\_SIZE, OFFSET);  if (bytes\_read == -1) {  close(fd);  handle\_error("Failed to read file for verification");  }  close(fd);  printf("File content on disk:\n%.\*s\n", (int)bytes\_read, file\_content);  printf("Program finished successfully.\n");  return 0;  } |



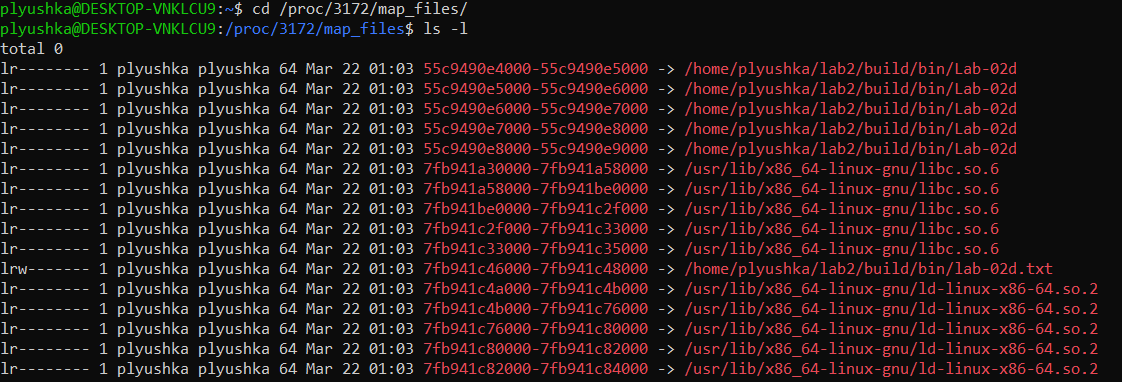
С помощью псевдофайловой системы «/proc» найти и изучить информацию о файле отображенном в память.

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.



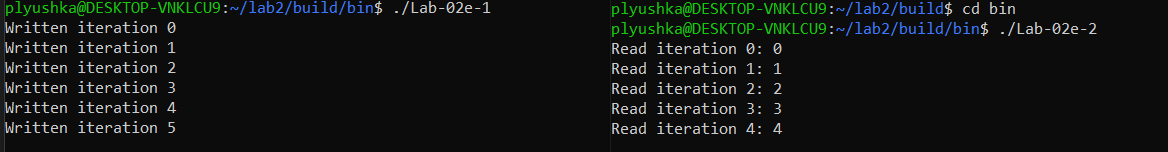
**Проект Lab-02e**: Разработать проект который состоит из двух приложений на языке C и функционально является аналогом приложения Lab-02c. Синхронизация должна быть реализована на «Семафорах».

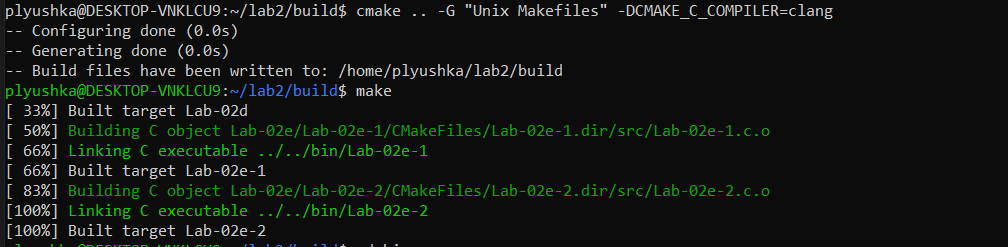
Приложение Lab-02e-1:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  #include <sys/mman.h>  #include <sys/stat.h>  #include <semaphore.h>  #include <string.h>  #define FILE\_SIZE (64 \* 1024) // 64 KiB  #define TOTAL\_SIZE (640 \* 1024) // 640 KiB  #define FILENAME "Lab-02"  #define SEM\_WRITE\_NAME "/lab02\_sem\_write"  #define SEM\_READ\_NAME "/lab02\_sem\_read"  void handle\_error(const char\* msg) {  perror(msg);  exit(EXIT\_FAILURE);  }  int main() {  int fd;  void\* mapped;  sem\_t\* sem\_write;  sem\_t\* sem\_read;  fd = open(FILENAME, O\_RDWR | O\_CREAT | O\_TRUNC, 0666);  if (fd == -1) {  handle\_error("Failed to open file");  }  if (ftruncate(fd, TOTAL\_SIZE) == -1) {  close(fd);  handle\_error("Failed to set file size");  }  mapped = mmap(NULL, TOTAL\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  if (mapped == MAP\_FAILED) {  close(fd);  handle\_error("Failed to map file to memory");  }  sem\_write = sem\_open(SEM\_WRITE\_NAME, O\_CREAT | O\_EXCL, 0666, 1);  if (sem\_write == SEM\_FAILED) {  munmap(mapped, TOTAL\_SIZE);  close(fd);  handle\_error("Failed to create write semaphore");  }  sem\_read = sem\_open(SEM\_READ\_NAME, O\_CREAT | O\_EXCL, 0666, 0);  if (sem\_read == SEM\_FAILED) {  sem\_close(sem\_write);  sem\_unlink(SEM\_WRITE\_NAME);  munmap(mapped, TOTAL\_SIZE);  close(fd);  handle\_error("Failed to create read semaphore");  }  for (int i = 0; i < 10; i++) {  sem\_wait(sem\_write);  int\* data = (int\*)((char\*)mapped + i \* FILE\_SIZE);  for (size\_t j = 0; j < FILE\_SIZE / sizeof(int); j++) {  data[j] = i;  }  printf("Written iteration %d\n", i);  sleep(1);  sem\_post(sem\_read);  }  munmap(mapped, TOTAL\_SIZE);  close(fd);  sem\_close(sem\_write);  sem\_close(sem\_read);  sem\_unlink(SEM\_WRITE\_NAME);  sem\_unlink(SEM\_READ\_NAME);  printf("Lab-02e-1 finished successfully.\n");  return 0;  } |

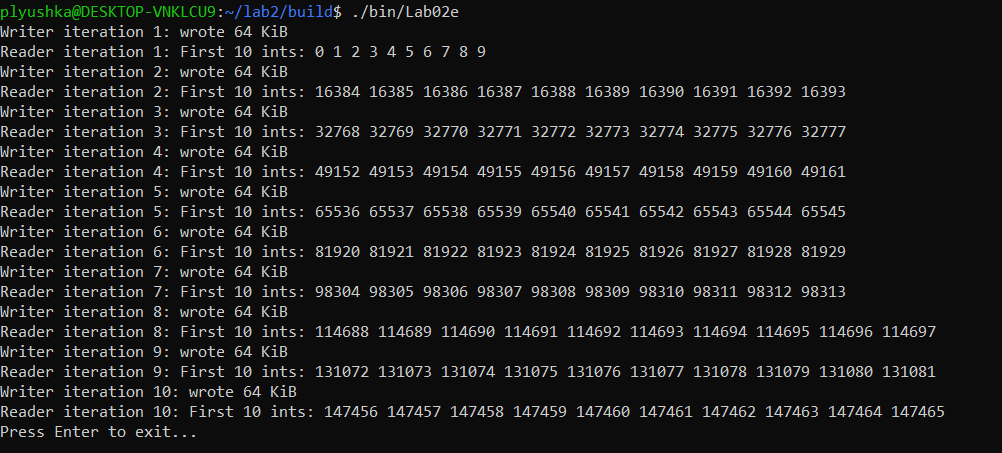
Приложение Lab-02e-2:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  #include <sys/mman.h>  #include <sys/stat.h>  #include <semaphore.h>  #include <string.h>  #define FILE\_SIZE (64 \* 1024) // 64 KiB  #define TOTAL\_SIZE (640 \* 1024) // 640 KiB  #define FILENAME "Lab-02"  #define SEM\_WRITE\_NAME "/lab02\_sem\_write"  #define SEM\_READ\_NAME "/lab02\_sem\_read"  void handle\_error(const char\* msg) {  perror(msg);  exit(EXIT\_FAILURE);  }  int main() {  int fd;  void\* mapped;  sem\_t\* sem\_write;  sem\_t\* sem\_read;  fd = open(FILENAME, O\_RDWR);  if (fd == -1) {  handle\_error("Failed to open file");  }  mapped = mmap(NULL, TOTAL\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  if (mapped == MAP\_FAILED) {  close(fd);  handle\_error("Failed to map file to memory");  }  sem\_write = sem\_open(SEM\_WRITE\_NAME, 0);  if (sem\_write == SEM\_FAILED) {  munmap(mapped, TOTAL\_SIZE);  close(fd);  handle\_error("Failed to open write semaphore");  }  sem\_read = sem\_open(SEM\_READ\_NAME, 0);  if (sem\_read == SEM\_FAILED) {  sem\_close(sem\_write);  munmap(mapped, TOTAL\_SIZE);  close(fd);  handle\_error("Failed to open read semaphore");  }  for (int i = 0; i < 10; i++) {  sem\_wait(sem\_read);  int\* data = (int\*)((char\*)mapped + i \* FILE\_SIZE);  printf("Read iteration %d: first element = %d\n", i, data[0]);  /\*printf("Read iteration %d:\n", i);  for (size\_t j = 0; j < FILE\_SIZE / sizeof(int); j++) {  printf("%d ", data[j]);  if ((j + 1) % 16 == 0) {  printf("\n");  }  }  printf("\n");\*/  sem\_post(sem\_write);  }  munmap(mapped, TOTAL\_SIZE);  close(fd);  sem\_close(sem\_write);  sem\_close(sem\_read);  printf("Lab-02e-2 finished successfully.\n");  return 0;  } |





|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/mman.h>  #include <semaphore.h>  #include <unistd.h>  #include <string.h>  #include <sys/wait.h>  #include <fcntl.h>  #define VIEW\_SIZE (64 \* 1024)  #define ITERATIONS 10  #define TOTAL\_SIZE (640 \* 1024)  #define INTS\_PER\_VIEW (VIEW\_SIZE / sizeof(int))  void handle\_error(const char\* message) {  perror(message);  exit(EXIT\_FAILURE);  }  int main() {  int\* shared\_mem = mmap(NULL, VIEW\_SIZE,  PROT\_READ | PROT\_WRITE,  MAP\_ANONYMOUS | MAP\_SHARED,  -1, 0);  if (shared\_mem == MAP\_FAILED)  handle\_error("mmap failed");  sem\_t\* sem\_producer = mmap(NULL, sizeof(sem\_t),  PROT\_READ | PROT\_WRITE,  MAP\_ANONYMOUS | MAP\_SHARED,  -1, 0);  sem\_t\* sem\_consumer = mmap(NULL, sizeof(sem\_t),  PROT\_READ | PROT\_WRITE,  MAP\_ANONYMOUS | MAP\_SHARED,  -1, 0);  if (sem\_producer == MAP\_FAILED || sem\_consumer == MAP\_FAILED)  handle\_error("semaphore mmap failed");  if (sem\_init(sem\_producer, 1, 1) == -1 ||  sem\_init(sem\_consumer, 1, 0) == -1)  handle\_error("sem\_init failed");  pid\_t pid = fork();  if (pid == -1) {  handle\_error("fork failed");  }  if (pid == 0) {  for (int i = 0; i < ITERATIONS; i++) {  sem\_wait(sem\_consumer);  printf("Reader iteration %d: First 10 ints: ", i + 1);  for (int j = 0; j < 10; j++) {  printf("%d ", shared\_mem[j]);  }  printf("\n");  sem\_post(sem\_producer);  }  munmap(shared\_mem, VIEW\_SIZE);  munmap(sem\_producer, sizeof(sem\_t));  munmap(sem\_consumer, sizeof(sem\_t));  exit(EXIT\_SUCCESS);  }  else {  int\* data = malloc(TOTAL\_SIZE);  if (!data) handle\_error("malloc failed");  for (size\_t i = 0; i < TOTAL\_SIZE / sizeof(int); i++) {  data[i] = i;  }  for (int i = 0; i < ITERATIONS; i++) {  sem\_wait(sem\_producer);  memcpy(shared\_mem,  data + i \* INTS\_PER\_VIEW,  VIEW\_SIZE);  printf("Writer iteration %d: wrote 64 KiB\n", i + 1);  sem\_post(sem\_consumer);  }  free(data);  wait(NULL);  munmap(shared\_mem, VIEW\_SIZE);  munmap(sem\_producer, sizeof(sem\_t));  munmap(sem\_consumer, sizeof(sem\_t));  printf("Press Enter to exit...\n");  getchar();  }  return 0;  } |



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?