

МИНОБРНАУКИ РОССИИ

---

Санкт-Петербургский государственный  
электротехнический университет «ЛЭТИ»

---

## **ОПЕРАЦИОННЫЕ СИСТЕМЫ**

Методические указания  
к лабораторным работам

Санкт-Петербург  
Издательство СПбГЭТУ «ЛЭТИ»

2016

УДК 004.454.9

Операционные системы: электронные методические указания к лабораторным работам / Сост.: А. В. Тимофеев. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2016. 46 с.

Содержат описания лабораторных работ по дисциплине «Операционные системы».

Предназначены для подготовки бакалавров по направлению 09.03.01 «Информатика и вычислительная техника», а также могут быть полезны инженерно-техническим работникам этой области знаний.

Рекомендовано методической комиссией факультета компьютерных технологий и информатики для использования в учебном процессе по дисциплине «Операционные системы»

© СПбГЭТУ «ЛЭТИ», 2016

## РАБОТА 1. ИССЛЕДОВАНИЕ ОБЪЕКТОВ WINDOWS

Целью лабораторной работы №1 является исследование объектных механизмов Win32.

### **Задание 1.1. Получение списка открытых объектов и изучение типов объектов**

Утилита *Process Explorer* заменяет *Диспетчер задач Windows*, отображая более подробную информацию о процессах и потоках, включая их родство, загруженные DLL и открытые дескрипторы объектов.

Консольная утилита *Handle* отображает информацию о дескрипторах объектов, имеющих у процессов системы. Если запустить утилиту без параметров командной строки, она выведет список всех процессов и все описатели файлов и именованных разделов, занятых этими процессами, разделяя данные о каждом процессе пунктирной линией. Для каждого процесса отображается имя, PID и имя учетной записи, от имени которой запущен процесс. Далее указываются описатели, принадлежащие этому процессу.

Для получения справочной информации по синтаксису командной строки утилиты можно набрать команду *handle -h*.

Следуйте следующим указаниям при выполнении задания:

1. Скачайте и распакуйте в каталог **c:\Tools** утилиту *Process Explorer* (<http://technet.microsoft.com/ru-ru/sysinternals/bb896653.aspx>).

2. Выполните запуск утилиты *Process Explorer* от имени администратора.

3. В интерфейсе *Process Explorer* выберите процесс для исследования и ознакомьтесь со списком объектов, принадлежащих выбранному процессу. Запишите в отчет результаты выполнения данного шага. На рисунке 1.1 в качестве примера выбран процесс *calc.exe* с идентификатором (PID) 4248<sub>10</sub>.

Обратите внимание, что значение дескриптора объекта всегда кратно 4, почему это так, мы рассмотрим в следующем задании.

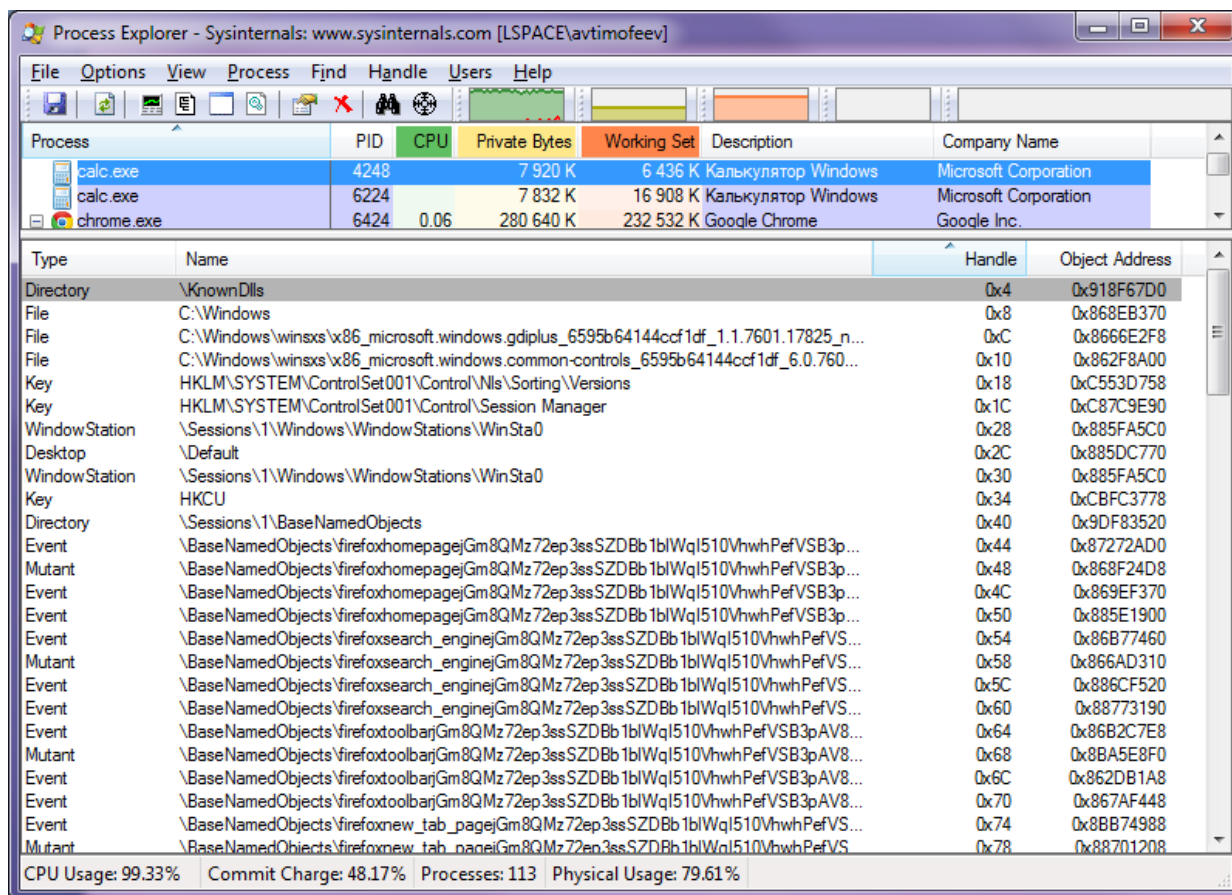


Рис. 1.1. Отображение объектов процесса в интерфейсе *Process Explorer*

4. Скачайте и распакуйте в каталог **c:\Tools** утилиту *Handle* (<http://technet.microsoft.com/ru-ru/sysinternals/bb896655>).

5. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools** и запустите утилиту *Handle.exe* со следующими параметрами: *handle -a -p <PID>*, где *PID* – идентификатор процесса, выбранного в пункте 3 этого задания. В результате Вы получите список объектов, принадлежащих выбранному процессу. Запишите в отчет результаты выполнения данного шага, сравните полученные сведения с результатами выполнения пункта 3. На рисунке 1.2 представлен перечень объектов процесса *calc.exe* с идентификатором (PID) 4248<sub>10</sub>.

```
Администратор: Командная строка

c:\Tools>handle -a -p 4248

Handle v3.51
Copyright (C) 1997-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

 4: Directory      \KnownDlls
 8: File (RW-)     C:\Windows
 C: File (RW-)     C:\Windows\winsxs\x86_microsoft.windows.gdiplus_6595b64144c
cf1df_1.1.7601.17825_none_72d273598668a06b
10: File (RW-)     C:\Windows\winsxs\x86_microsoft.windows.common-controls_659
5b64144cf1df_6.0.7601.17514_none_41e6975e2bd6f2b2
14: ALPC Port
18: Key           HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions
1C: Key           HKLM\SYSTEM\ControlSet001\Control\Session Manager
20: EtwRegistration
24: Event
28: WindowStation \Sessions\1\Windows\WindowStations\WinSta0
2C: Desktop       \Default
30: WindowStation \Sessions\1\Windows\WindowStations\WinSta0
34: Key           HKCU
38: EtwRegistration
3C: EtwRegistration
40: Directory      \Sessions\1\BaseNamedObjects
44: Event         \BaseNamedObjects\firefoxhomepagejGm8QMz72ep3ssSZDBb1b1WqIS
10VhwhPefVSB3pAV8=_2.6.1249.132
```

Рис. 1.2. Отображение объектов процесса с использованием утилиты *Handle*

6. Запустите утилиту *Handle.exe* со следующими параметрами: *handle -s -p <PID>*, где *PID* – идентификатор, выбранного в пункте 3 процесса. В результате Вы получите сводный список с количеством объектов разного типа, принадлежащих выбранному процессу (см. рис. 1.3). Запишите в отчет результаты выполнения данного шага, с помощью Интернет выясните назначение этих типов объектов и прокомментируйте их в отчете.

```
Администратор: Командная строка

c:\Tools>handle -s -p calc

Handle v3.51
Copyright (C) 1997-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

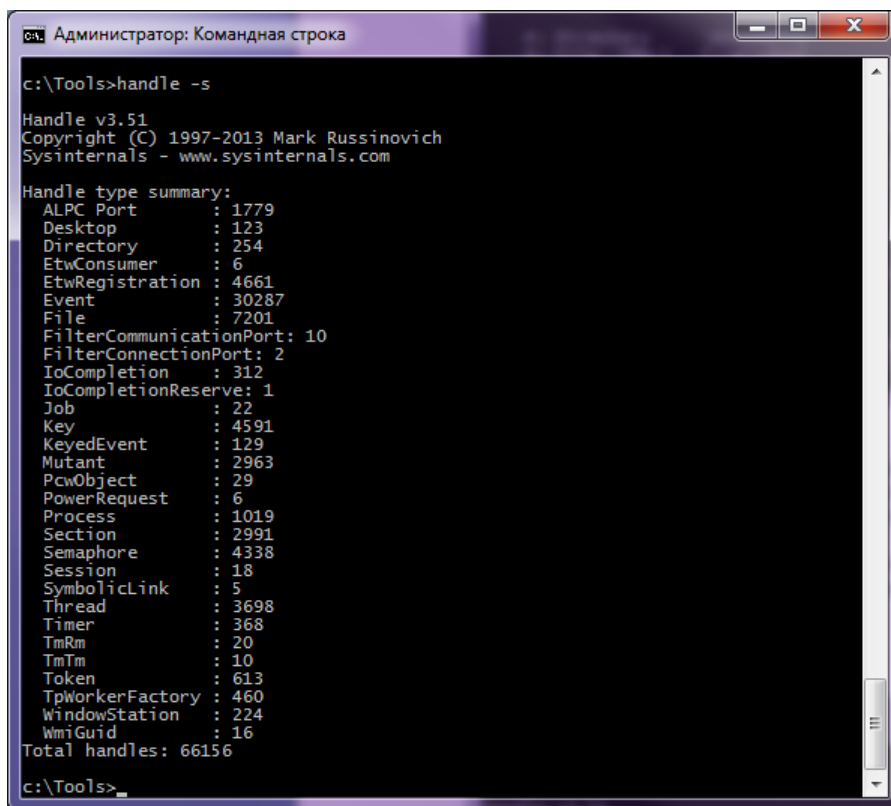
Handle type summary:
ALPC Port      : 6
Desktop        : 2
Directory      : 4
EtwRegistration : 40
Event          : 130
File           : 56
Key            : 18
Mutant         : 34
Section        : 58
Semaphore      : 16
Thread         : 6
Timer          : 2
WindowStation  : 4
Total handles: 376

c:\Tools>
```

Рис. 1.3. Получение сводного списка объектов процесса с использованием утилиты *Handle*

7. Запустите утилиту *Handle.exe* со следующими параметрами: *handle -s*. В результате Вы получите сводный список с количеством объектов разного

типа для всех процессов вычислительной системы (см. рис. 1.4). Запишите в отчет результаты выполнения данного шага, поясните назначение представленных в отчете типов объектов.



```
c:\Tools>handle -s

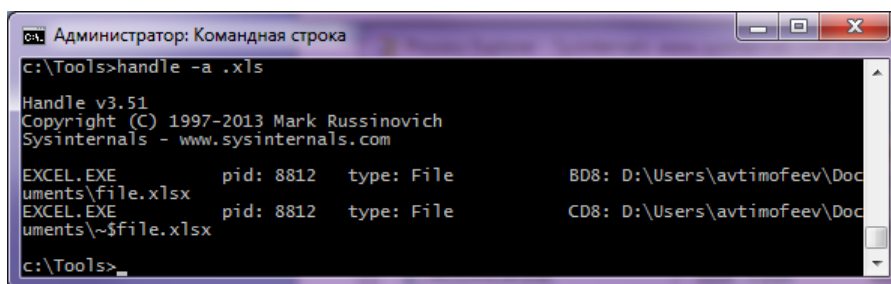
Handle v3.51
Copyright (C) 1997-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

Handle type summary:
ALPC Port      : 1779
Desktop       : 123
Directory     : 254
EtwConsumer    : 6
EtwRegistration : 4661
Event         : 30287
File          : 7201
FilterCommunicationPort: 10
FilterConnectionPort: 2
IoCompletion   : 312
IoCompletionReserve: 1
Job           : 22
Key           : 4591
KeyedEvent    : 129
Mutant        : 2963
PcwObject     : 29
PowerRequest  : 6
Process       : 1019
Section       : 2991
Semaphore     : 4338
Session       : 18
SymbolicLink  : 5
Thread        : 3698
Timer         : 368
TmRm         : 20
TmTm         : 10
Token         : 613
TpWorkerFactory : 460
WindowStation : 224
WmiGuid       : 16
Total handles: 66156

c:\Tools>
```

Рис. 1.4. Получение сводного списка объектов всех объектов системы

8. При помощи *Handle.exe* можно закрывать дескрипторы объектов, открытых процессом, не завершая сам процесс. В качестве примера рассмотрим ситуацию, когда в приложении *Excel* открыт файл *test.xlsx*. Проверим дескрипторы открытых в системе файлов с расширением *.xls\** с помощью команды *handle -a .xls*. В результате мы получим перечень из двух объектов типа *File* (см. рис. 1.5).



```
c:\Tools>handle -a .xls

Handle v3.51
Copyright (C) 1997-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

EXCEL.EXE      pid: 8812   type: File      BD8: D:\Users\avtimofeev\Doc
uments\file.xlsx
EXCEL.EXE      pid: 8812   type: File      CD8: D:\Users\avtimofeev\Doc
uments\~$file.xlsx

c:\Tools>
```

Рис. 1.5. Получение дескрипторов открытых файлов с расширением *.xls\**

Закроем принудительно дескриптор одного из этих объектов с помощью команды: `handle -c CD8 -p 8812 -y` (см. рис. 1.6).

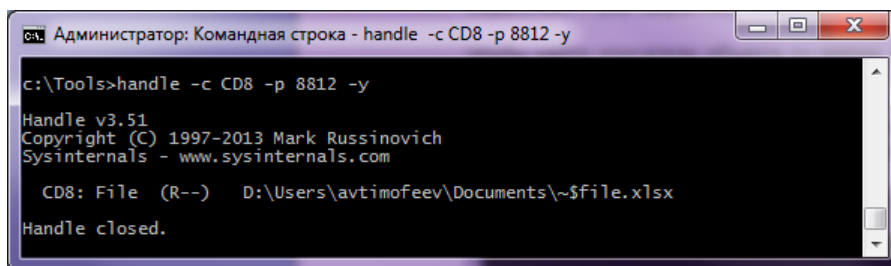


Рис. 1.6. Заккрытие дескриптора объекта

Затем снова проверим дескрипторы открытых в системе файлов с расширением `.xls*` с помощью команды: `handle -a .xls` (см. рис. 1.7).

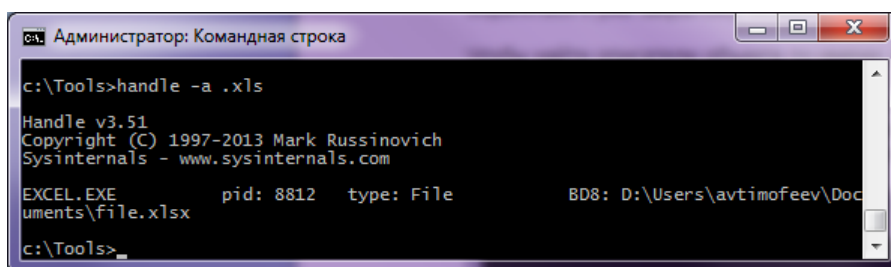


Рис. 1.7. Получение дескрипторов открытых файлов с расширением `.xls`

Повторите подобный эксперимент и запишите в отчет результаты.

**Внимание:** Поскольку процесс, владеющий дескриптором объекта, не знает о том, что этот дескриптор закрыт, использование этой функции может привести к повреждению данных или сбою в приложении; закрытие дескриптора в системном процессе или критически важном процессе пользовательского режима, таком как `csrss`, может привести к сбою в системе. Кроме того, при последующем распределении ресурсов тем же процессом может быть получено старое значение дескриптора, так как оно больше не используется. В этом случае процесс может получить доступ не к тому объекту, который ожидает.

9. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## Задание 1.2. Изучение хранения информации об объектах процесса

В рамках этого задания Вам будет предложено использовать программу *LiveKD*, которая позволяет запускать отладчик ядра Microsoft Kd, входящий в Windows Driver Kit (WDK). *LiveKD* позволяет выполнять просмотр моментального снимка работающей локальной системы, не перезагружая систему в режим отладки. Поскольку *LiveKd* получает дампы физической памяти, возможны ситуации, когда структуры данных находятся в процессе изменения системой и не являются согласованными. Запускаясь, отладчик каждый раз начинает работать со свежим снимком системы. Для его обновления выполните выход из отладчика (команда q), и затем ответьте утвердительно на вопрос о перезапуске *LiveKd*.

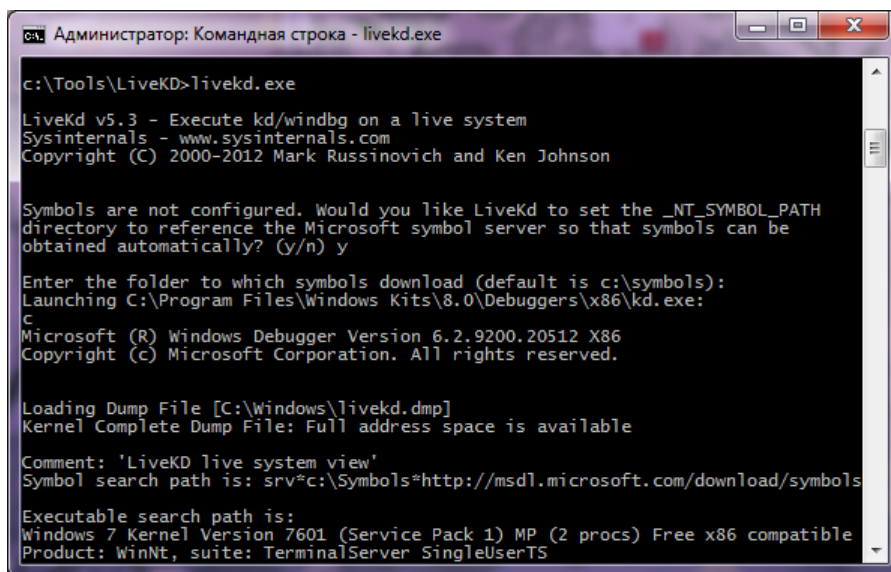
Для получения справки по той или иной команде отладчика, Вы можете просто набрать команду *.hh <имя команды>*, после чего Вам будет предложен для изучения соответствующий раздел файла справочной системы. Если вдруг Вам потребуется прервать работу некоторой команды, то Вы можете нажать Ctrl+Break, после этого Вам будет предложено перезапустить *LiveKD* или выйти из отладчика.

Следуйте следующим указаниям при выполнении задания:

1. Загрузите пакет Windows Driver Kit (WDK), находящийся по адресу <http://msdn.microsoft.com/en-US/windows/hardware/hh852362>
2. Установите на компьютер пакет Windows Driver Kit (WDK) с настройками по умолчанию.
3. Загрузите программу *LiveKd*, размещенную по адресу <http://technet.microsoft.com/ru-ru/sysinternals/bb897415.aspx>.
4. Извлеките из архива утилиту *LiveKd.exe* и поместите в каталог **c:\Tools\LiveKD**.
5. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту *LiveKd.exe*. В процессе запуска



программы Вам могут быть заданы некоторые вопросы, касающиеся настроек запуска, отвечайте на них утвердительно (см. рис. 1.8).



```
Администратор: Командная строка - livekd.exe

c:\Tools\LiveKD>livekd.exe

LiveKd v5.3 - Execute kd/windbg on a live system
Sysinternals - www.sysinternals.com
Copyright (C) 2000-2012 Mark Russinovich and Ken Johnson

Symbols are not configured. Would you like LiveKd to set the _NT_SYMBOL_PATH
directory to reference the Microsoft symbol server so that symbols can be
obtained automatically? (y/n) y

Enter the folder to which symbols download (default is c:\symbols):
Launching C:\Program Files\Windows Kits\8.0\Debuggers\x86\kd.exe:
C:
Microsoft (R) Windows Debugger Version 6.2.9200.20512 X86
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\Windows\livekd.dmp]
Kernel Complete Dump File: Full address space is available

Comment: 'LiveKD live system view'
Symbol search path is: srv*c:\symbols=http://msdl.microsoft.com/download/symbols

Executable search path is:
Windows 7 Kernel Version 7601 (Service Pack 1) MP (2 procs) Free x86 compatible
Product: WinNT, suite: TerminalServer SingleUserTS
```

Рис. 1.8. Запуск утилиты *LiveKd.exe*

6. С помощью команды *!handle* можно получить информацию об открытых дескрипторах объектов. Команда имеет следующую форму:

*!handle* <индекс\_дескриптора> <флаги>  
<идентификатор\_процесса>.

Индекс дескриптора определяет элемент в таблице дескрипторов (0 – вывод всех дескрипторов). Вы можете указывать флаги, являющиеся битовыми масками, где бит 0 означает, что нужно вывести лишь информацию из элемента таблицы, бит 1 – показать не только используемые, но и свободные дескрипторы, а бит 2 – сообщить информацию об объекте, на который ссылается дескриптор. Следующая команда выводит полную информацию о таблице дескрипторов в процессе с идентификатором *0x1098* (*4248<sub>10</sub>*) (см. рис 1.9).

Команда *!handle 284 3 1098* позволяет получить подробную информацию об выбранном объекте (*0x284*) заданного процесса (*0x1098*) (см. рис. 1.10).

```

Администратор: Командная строка - LiveKD\livekd.exe
0: kd> !handle 0 3 1098

Searching for Process with Cid == 1098
PROCESS 866aa7d8 SessionId: 1 Cid: 1098 Peb: 7ffd3000 ParentCid: 162c
DirBase: bd59f300 ObjectTable: cc202c30 HandleCount: 188.
Image: calc.exe

Handle table at cc202c30 with 188 entries in use

0004: Object: 918f67d0 GrantedAccess: 00000003 Entry: 9cf99008
Object: 918f67d0 Type: (86106610) Directory
ObjectHeader: 918f67b8 (new version)
HandleCount: 107 PointerCount: 145
Directory Object: 8d405ed0 Name: KnownDlls

Hash Address Type Name
----
00 9a386540 Section IMAGEHLP.dll
9a3aac10 Section gdi32.dll
9de08d58 Section kernelbase.dll
02 9a395fd8 Section NORMALIZ.dll
03 9a3a5ee0 Section ole32.dll
9a37adb8 Section URLMON.dll
04 9a378040 Section USP10.dll
05 9a3f65f8 Section DEVOBJ.dll
06 9a3a5ec0 Section SHELL32.dll
9a3f6498 Section CFGMGR32.dll
9a3b1b18 Section WLDAP32.dll
09 9a381fd8 Section user32.dll
14 9a3f35a0 Section MSASN1.dll
16 918e4158 SymbolicLink KnownDllPath
9a3fe110 Section COMCTL32.dll
17 9a394cf8 Section PSAPI.DLL
9a3f4e08 Section CRYPT32.dll
18 9a378a28 Section advapi32.dll
9a3d1b50 Section OLEAUT32.dll
19 9a3d1590 Section SHLWAPI.dll
9a3957f0 Section IERTUTIL.dll
919a7da0 Section ntdll.dll
20 9a3fdf10 Section WS2_32.dll
21 9a37ae40 Section LPK.dll
22 9a3a5f70 Section sechost.dll
23 9a386870 Section COMDLG32.dll
24 9a3b4750

```

Рис. 1.9. Вывод полной информации о таблице дескрипторов

```

Администратор: Командная строка - LiveKD\livekd.exe
0: kd> !handle 284 3 1098

Searching for Process with Cid == 1098
PROCESS 866aa7d8 SessionId: 1 Cid: 1098 Peb: 7ffd3000 ParentCid: 162c
DirBase: bd59f300 ObjectTable: cc202c30 HandleCount: 188.
Image: calc.exe

Handle table at cc202c30 with 188 entries in use

0284: Object: 86457828 GrantedAccess: 001fffff Entry: 9cf99508
Object: 86457828 Type: (8610bc18) Thread
ObjectHeader: 86457810 (new version)
HandleCount: 2 PointerCount: 4

0: kd>

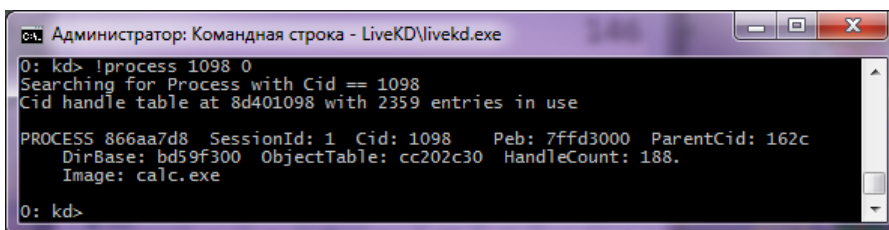
```

Рис. 1.10. Вывод подробной информации о выбранном объекте

Выполните с помощью команды *!handle* вывод списка объектов, принадлежащих процессу, выбранному в пункте 3 прошлого задания. Запишите в отчет результаты выполнения данного шага, сравните полученные сведения с данными из пункта 3 прошлого задания.

7. С помощью команды *!process* можно получить информацию о всех процессах вычислительной системы. Команда *!process 1098 0* позволяет

получить краткую информацию о выбранном процессе (0x1098), в том числе получить адрес его таблицы объектов (ObjectTable) (см. рис. 1.11).



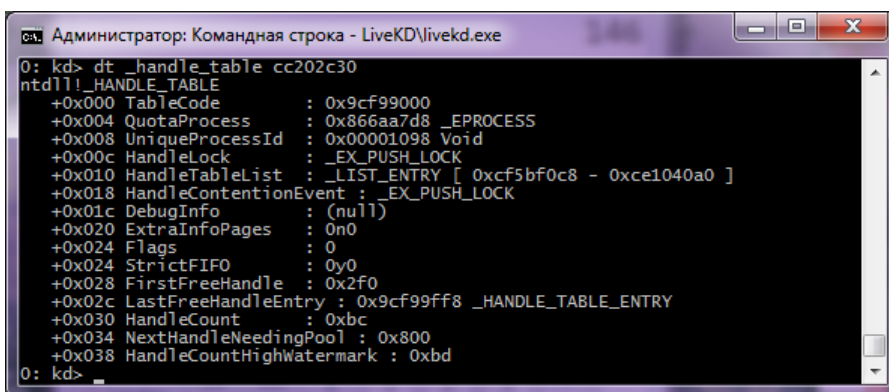
```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> !process 1098 0
Searching for Process with Cid == 1098
Cid handle table at 8d401098 with 2359 entries in use

PROCESS 866aa7d8 SessionId: 1 Cid: 1098 Peb: 7ffd3000 ParentCid: 162c
DirBase: bd59f300 ObjectTable: cc202c30 HandleCount: 188.
Image: calc.exe
0: kd>
```

Рис. 1.11. Вывод краткой информации о процессе

Получите с помощью команды *!process* информацию о процессе, выбранном в пункте 3 прошлого задания. Запишите в отчет результаты выполнения данного шага.

8. Чтобы проанализировать поля таблицы объектов процесса, следует воспользоваться командой *dt \_handle\_table <адрес\_таблицы>* (см. рис. 1.12).



```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> dt _handle_table cc202c30
ntdll!_HANDLE_TABLE
+0x000 TableCode           : 0x9cf99000
+0x004 QuotaProcess        : 0x866aa7d8 _EPROCESS
+0x008 UniqueProcessId    : 0x00001098 Void
+0x00c HandleLock         : _EX_PUSH_LOCK
+0x010 HandleTableList    : _LIST_ENTRY [ 0xcf5bf0c8 - 0xce1040a0 ]
+0x018 HandleContentionEvent : _EX_PUSH_LOCK
+0x01c DebugInfo          : (null)
+0x020 ExtraInfoPages     : 0n0
+0x024 Flags              : 0
+0x028 StrictFIFO         : 0y0
+0x02c FirstFreeHandle    : 0x2f0
+0x030 LastFreeHandleEntry : 0x9cf99ff8 _HANDLE_TABLE_ENTRY
+0x034 HandleCount        : 0xbc
+0x038 NextHandleNeedingPool : 0x800
+0x03c HandleCountHighWatermark : 0xbd
0: kd>
```

Рис. 1.12. Вывод таблицы объектов процесса

Для дальнейшего анализа таблицы объектов наибольший интерес представляет поле *TableCode*. Поле *TableCode* структуры *HANDLE\_TABLE* в двух младших битах *Attr* содержит данные о числе уровней таблицы (от одного до трех), остальные биты *TableCode* представляют собой указатель на таблицу первого уровня.

Например, если *Attr* равен нулю, то число таблиц равно единице, следовательно, первый уровень содержит элементы *TABLE\_ENTRY* (то есть *TableCode* указывает на массив *TABLE\_ENTRY*). Если *Attr* равен единице, то таблиц две, и значит, *TableCode* указывает на промежуточную (вторую)

таблицу, каждый элемент которой и включает указатели на TABLE\_ENTRY и т.д.

Получите с помощью команды *dt \_handle\_table* информацию о таблице объектов выбранного ранее процесса. Запишите в отчет результаты выполнения данного шага.

9. Проанализируем с помощью команды *dd <физический\_адрес>* первые 64 16-разрядных слова, размещенные по адресу одноуровневой таблицы объектов процесса. Обратим внимание, что каждому объекту ставится в соответствие четыре 16-разрядных слова (см. рис. 1.13). В качестве идентификатора объекта используется смещение элемента объекта в 16-разрядных словах относительно начала таблицы. Элемент 0 не используется для хранения объекта. Поэтому первый объект в таблице имеет идентификатор 4, второй – 8 и т.д.

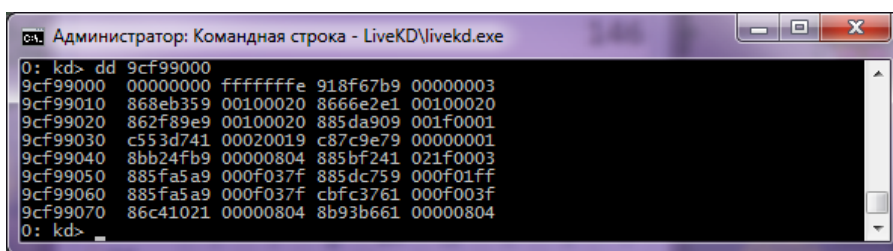


Рис. 1.13. Получение таблицы дескрипторов процесса

Структура элемента таблицы дескрипторов представлена на рисунке 1.14.

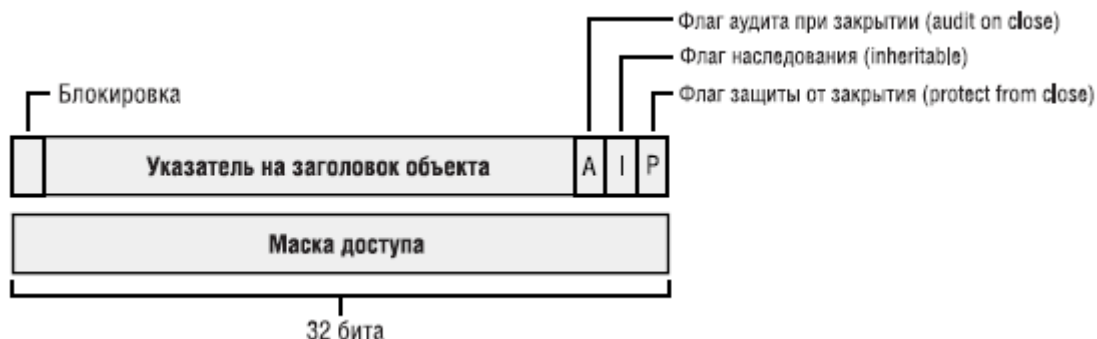


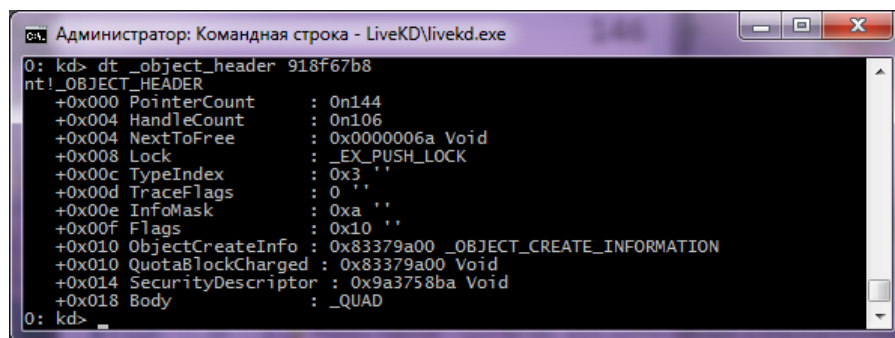
Рис. 1.14. Структура элемента таблицы дескрипторов

Чтобы получить адрес заголовка объекта необходимо выполнить следующую операцию над младшими 32 битами элемента таблицы

дескрипторов: ( $\langle \text{младшие 32 бита} \rangle \mid 0x80000000$ )  $\& 0xffffffff$ . Так в рассмотренном выше примере в таблице объектов под дескриптором 4 хранится значение 918F67B9 00000003, в этом случае адрес заголовка объекта будет:  $0x918F67B9 \mid 0x80000000$ )  $\& 0xffffffff = 0x918F67B8$ .

Получите с помощью команды *dd* содержимое первых записей таблицы объектов выбранного ранее процесса, определите физические адреса объектов с дескрипторами 4, 8, 12. Запишите в отчет результаты выполнения данного шага.

10. Для чтения заголовка объекта по его адресу необходимо воспользоваться командой *dt \_object\_header <физический\_адрес>*. Например, на рисунке 1.15 приведен заголовок объекта с дескриптором 4 из нашего примера.



```
0: kd> dt _object_header 918f67b8
nt!_OBJECT_HEADER
+0x000 PointerCount      : 0n144
+0x004 HandleCount      : 0n106
+0x004 NextToFree       : 0x0000006a Void
+0x008 Lock              : _EX_PUSH_LOCK
+0x00c TypeIndex        : 0x3
+0x00d TraceFlags       : 0
+0x00e InfoMask         : 0xa
+0x00f Flags             : 0x10
+0x010 ObjectCreateInfo : 0x83379a00 _OBJECT_CREATE_INFORMATION
+0x010 QuotaBlockCharged : 0x83379a00 Void
+0x014 SecurityDescriptor : 0x9a3758ba Void
+0x018 Body              : _QUAD
```

Рис. 1.15. Чтение заголовка объекта

Получите с помощью команды *dt \_object\_header* содержимое заголовков объектов с дескрипторами 4, 8, 12. Запишите в отчет результаты выполнения данного шага.

11. Чтобы получить описание самого объекта необходимо обратиться к полю *Body* заголовка объекта по смещению 0x18. Для этого необходимо использовать команду *!object*, пример приведен на рисунке 1.16.

Получите с помощью команды *!object* описание объектов с дескрипторами 4, 8, 12. Запишите в отчет результаты выполнения данного шага.

```

0: kd> !object 918f67b8+18
Object: 918f67d0 Type: (86106610) Directory
ObjectHeader: 918f67b8 (new version)
HandleCount: 106 PointerCount: 144
Directory Object: 8d405ed0 Name: KnownDlls

Hash Address Type Name
----
00 9a386540 Section IMAGEHLP.dll
01 9a3aac10 Section gdi32.dll
02 9de08d58 Section kernelbase.dll
03 9a395fd8 Section NORMALIZ.dll
04 9a3a5ee0 Section ole32.dll
05 9a37adb8 Section URLMON.dll
06 9a378040 Section USP10.dll
07 9a3f65f8 Section DEVOBJ.dll
08 9a3a5ec0 Section SHELL32.dll
09 9a3b1b18 Section WLDAP32.dll
10 9a3f6498 Section CFGMGR32.dll
11 9a381fd8 Section user32.dll
12 9a3f35a0 Section MSASN1.dll
13 918e4158 SymbolicLink KnownDllPath
14 9a3fe110 Section COMCTL32.dll
15 9a394cf8 Section PSAPI.DLL
16 9a3f4e08 Section CRYPT32.dll
17 9a378a28 Section advapi32.dll
18 9a3d1b50 Section OLEAUT32.dll
19 9a3d1590 Section SHLWAPI.dll
20 9a3957f0 Section IERTUTIL.dll
21 919a7da0 Section ntdll.dll
22 9a3fdff0 Section WS2_32.dll
23 9a37ae40 Section LPK.dll
24 9a3a5f70 Section sechost.dll
25 9a386870 Section COMDLG32.dll
26 9a3b4750 Section difxapi.dll
27 9a3f8e78 Section Setupapi.dll
28 9a378178 Section MSCTF.dll
29 9a38e890 Section WININET.dll
30 9a37ed38 Section IMM32.dll
31 9a3f4ef0 Section WINTRUST.dll
32 9a3bbc58 Section MSVCRT.dll
33 9a3a5fd8 Section rpcrt4.dll
34 9a3b1fd8 Section clbcatq.dll
35 9a3c3518 Section kernel32.dll
36 9a3f3470 Section NSI.dll
0: kd>

```

Рис. 1.16. Чтение описания объекта

12. Для получения сведений о типе объекта необходимо использовать команду *dt \_object\_type*, в качестве параметра которой передается значение поля *Type*, полученное после использования команды *!object* (см. рис. 1.17). Структура типа включает имя типа объекта, счетчики активных объектов этого типа, а также счетчики пикового числа дескрипторов и объектов данного типа. В поле *TypeInfo* хранится указатель на структуру данных, в которой содержатся атрибуты, общие для всех объектов этого типа, а также указатели на методы типа.

```

0: kd> dt _object_type 86106610
ntdll!_OBJECT_TYPE
+0x000 TypeList      : _LIST_ENTRY [ 0x86106610 - 0x86106610 ]
+0x008 Name          : _UNICODE_STRING "Directory"
+0x010 DefaultObject : 0x83386860 Void
+0x014 Index         : 0x3
+0x018 TotalNumberOfObjects : 0x3a
+0x01c TotalNumberOfHandles : 0xee
+0x020 HighWaterNumberOfObjects : 0x40
+0x024 HighWaterNumberOfHandles : 0x133
+0x028 TypeInfo      : _OBJECT_TYPE_INITIALIZER
+0x078 TypeLock      : _EX_PUSH_LOCK
+0x07c Key           : 0x65726944
+0x080 CallbackList  : _LIST_ENTRY [ 0x86106690 - 0x86106690 ]
0: kd>

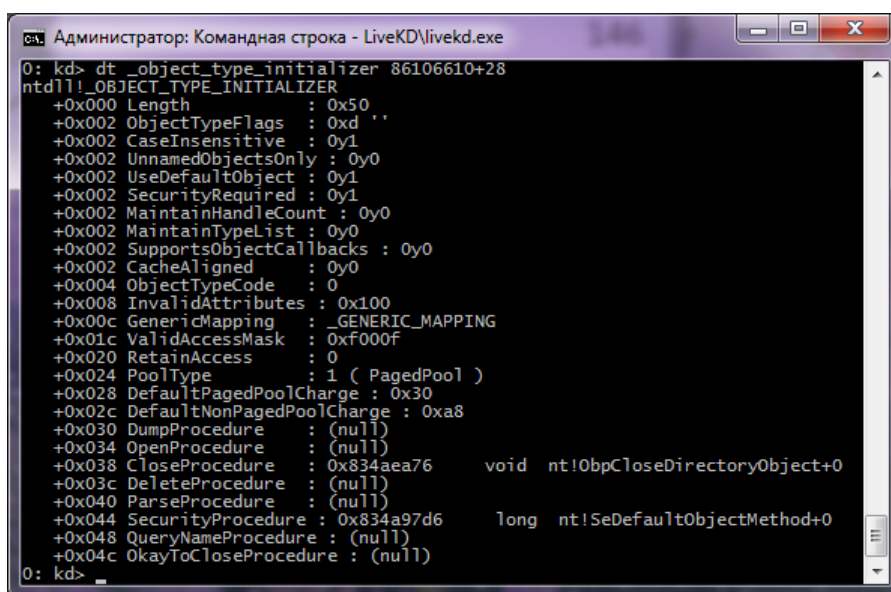
```

Рис. 1.17. Чтение сведений о типе объекта



Получите с помощью команды *dt \_object\_type* сведений о типах для объектов с дескрипторами 4, 8, 12. Запишите в отчет результаты выполнения данного шага.

13. Выполним анализ структуры данных, в которой содержатся атрибуты, общие для всех объектов типа, а также указатели на методы типа. Для вывода содержимого поля используем команду *dt \_object\_type\_initializer* <физический адрес>, не забываем, что поле *TypeInfo* имеет смещение 0x28 относительно начала структуры сведений о типе объекта (см. рис. 1.18).



```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> dt _object_type_initializer 86106610+28
ntdll!_OBJECT_TYPE_INITIALIZER
+0x000 Length : 0x50
+0x002 ObjectTypeInfoFlags : 0xd
+0x002 CaseInsensitive : 0y1
+0x002 UnnamedObjectsOnly : 0y0
+0x002 UseDefaultObject : 0y1
+0x002 SecurityRequired : 0y1
+0x002 MaintainHandleCount : 0y0
+0x002 MaintainTypeList : 0y0
+0x002 SupportsObjectCallbacks : 0y0
+0x002 CacheAligned : 0y0
+0x004 ObjectTypeCode : 0
+0x008 InvalidAttributes : 0x100
+0x00c GenericMapping : _GENERIC_MAPPING
+0x01c ValidAccessMask : 0xf000f
+0x020 RetainAccess : 0
+0x024 PoolType : 1 ( PagedPool )
+0x028 DefaultPagedPoolCharge : 0x30
+0x02c DefaultNonPagedPoolCharge : 0xa8
+0x030 DumpProcedure : (null)
+0x034 OpenProcedure : (null)
+0x038 CloseProcedure : 0x834aea76 void nt!ObpCloseDirectoryObject+0
+0x03c DeleteProcedure : (null)
+0x040 ParseProcedure : (null)
+0x044 SecurityProcedure : 0x834a97d6 long nt!SeDefaultObjectMethod+0
+0x048 QueryNameProcedure : (null)
+0x04c OkayToCloseProcedure : (null)
0: kd>
```

Рис. 1.18. Чтение структуры с атрибутами общими для всех объектов  
выбранного типа

Получите с помощью команды *dt \_object\_type\_initializer* общих атрибутов и методов типов для объектов с дескрипторами 4, 8, 12. Запишите в отчет результаты выполнения данного шага.

14. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## РАБОТА 2. ИССЛЕДОВАНИЕ СТРУКТУР ДАННЫХ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ В WINDOWS

Целью лабораторной работы №2 является исследование структур данных Windows, используемые для обеспечения безопасности.

### Задание 2.1. Определение идентификатора защиты SID текущего пользователя

В рамках этого задания Вы должны будете научиться определять идентификатор защиты SID текущего пользователя с помощью утилит *Process Explorer* и *PsGetSid*.

Утилита *PsGetSid* специально предназначена для получения SID разных учетных записей. Данная утилита входит в набор *PsTools* и её можно скачать с сайта *Sysinternals*.

Следуйте следующим указаниям при выполнении задания:

1. Выполните запуск утилиты *Process Explorer* от имени администратора.

2. В интерфейсе *Process Explorer* выберите процесс для исследования и нажмите кнопку Properties (см. рис. 2.1). В появившемся диалоговом окне выберите закладку **Security**, на которой отображается информация о маркера процесса (базовое имя пользователя, под записью которого работает процесс; группы, в которые входит эта запись, и ее привилегии в системе) (см. рис. 2.2). При выборе группы в списке **Group** под списком отображается идентификатор защиты (SID) выбранной группы.

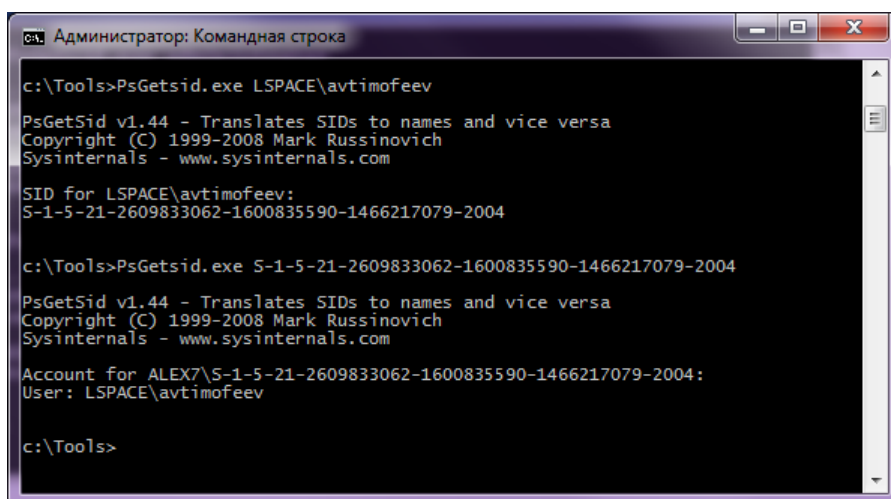




На рисунке вверху красным цветом выделен идентификатор безопасности (SID) пользователя – владельца процесса. SID представляет собой уникальное значение переменной длины, используемое в операционных системах Windows для идентификации участника безопасности или группы безопасности.

3. Скачайте и распакуйте в каталог **c:\Tools** комплект утилит *PsTools* (<http://technet.microsoft.com/ru-ru/sysinternals/bb897417>).

4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools** и запустите утилиту *PsGetSid.exe*. В качестве параметра утилиты можно указать либо имя учетной записи, либо SID. На рисунке 2.3 продемонстрированы оба варианта.



```
Администратор: Командная строка

c:\Tools>PsGetsid.exe LSPACE\avtimofeev

PsGetSid v1.44 - Translates SIDs to names and vice versa
Copyright (C) 1999-2008 Mark Russinovich
Sysinternals - www.sysinternals.com

SID for LSPACE\avtimofeev:
S-1-5-21-2609833062-1600835590-1466217079-2004

c:\Tools>PsGetsid.exe S-1-5-21-2609833062-1600835590-1466217079-2004

PsGetSid v1.44 - Translates SIDs to names and vice versa
Copyright (C) 1999-2008 Mark Russinovich
Sysinternals - www.sysinternals.com

Account for ALEX7\S-1-5-21-2609833062-1600835590-1466217079-2004:
User: LSPACE\avtimofeev

c:\Tools>
```

Рис. 2.3. Получение имени учетной записи и SID

Удостоверьтесь, что SID полностью совпадают в первом и втором способах.

5. Подготовьте итоговый отчет с развернутыми выводами по заданию.

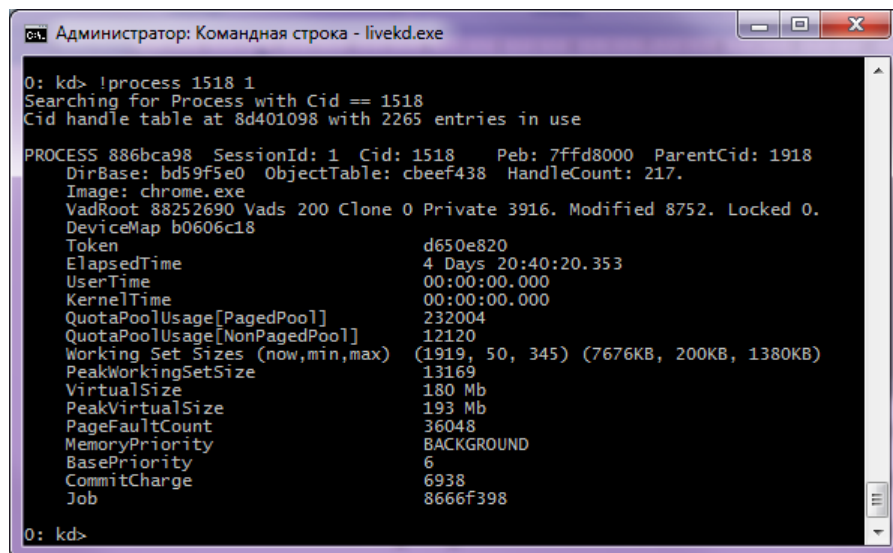
## Задание 2.2. Исследование маркера доступа (access token)

Следуйте следующим указаниям при выполнении задания:

1. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

2. Вызовите команду **!process 0 0** для вывода краткого списка процессов системы, выберите процесс для дальнейшего изучения и запишите его идентификатор (тоже самое Вы можете выполнить с помощью утилиты **Process Explorer**). В нашем случае для дальнейшего анализа выбран процесс **chrome.exe** с идентификатором **1518**. Ознакомьтесь с результатами выполнения данного шага, запишите их в отчет, выберите процесс для дальнейшего изучения.

3. Выполните команду **!process <идентификатор процесса> 1**, в этом случае Вам отобразится подробная информация о выбранном процессе. Для дальнейших исследований нас будет интересовать поле **Token**, значение которого равно **dd650e820** (см. рис. 2.4). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.



```
0: kd> !process 1518 1
Searching for Process with Cid == 1518
Cid handle table at 8d401098 with 2265 entries in use

PROCESS 886bca98 SessionId: 1 Cid: 1518 Peb: 7ffd8000 ParentCid: 1918
DirBase: bd59f5e0 ObjectTable: cbeef438 HandleCount: 217.
Image: chrome.exe
VadRoot 88252690 Vads 200 Clone 0 Private 3916. Modified 8752. Locked 0.
DeviceMap b0606c18
Token                                dd650e820
ElapsedTime                          4 Days 20:40:20.353
UserTime                             00:00:00.000
KernelTime                           00:00:00.000
QuotaPoolUsage[PagedPool]            232004
QuotaPoolUsage[NonPagedPool]        12120
Working Set Sizes (now,min,max)      (1919, 50, 345) (7676KB, 200KB, 1380KB)
PeakWorkingSetSize                   13169
VirtualSize                          180 Mb
PeakVirtualSize                      193 Mb
PageFaultCount                       36048
MemoryPriority                        BACKGROUND
BasePriority                          6
CommitCharge                         6938
Job                                  8666f398

0: kd>
```

Рис. 2.4. Получение маркера доступа процесса

4. Ознакомьтесь с помощью команды **dt** с содержимым структуры **\_TOKEN**, по адресу, определенному в предыдущем пункте, запишите результаты в отчет (см. рис. 2.5).

```

0: kd> dt _token 0xd650e820
nt!_TOKEN
+0x000 TokenSource      : _TOKEN_SOURCE
+0x010 TokenId          : _LUID
+0x018 AuthenticationId : _LUID
+0x020 ParentTokenId    : _LUID
+0x028 ExpirationTime   : _LARGE_INTEGER 0x7fffffff`ffffffff
+0x030 TokenLock        : 0x86713730 _ERESOURCE
+0x034 ModifiedId       : _LUID
+0x040 Privileges        : _SEP_TOKEN_PRIVILEGES
+0x058 AuditPolicy       : _SEP_AUDIT_POLICY
+0x074 SessionId        : 1
+0x078 UserAndGroupCount : 0xf
+0x07c RestrictedSidCount : 1
+0x080 VariableLength    : 0x210
+0x084 DynamicCharged    : 0x400
+0x088 DynamicAvailable : 0
+0x08c DefaultOwnerIndex : 0
+0x090 UserAndGroups     : 0xd650e9fc _SID_AND_ATTRIBUTES
+0x094 RestrictedSids     : 0xd650ea74 _SID_AND_ATTRIBUTES
+0x098 PrimaryGroup      : 0xe7970078 Void
+0x09c DynamicPart       : 0xe7970078 -> 0x501
+0x0a0 DefaultDacl       : 0xe7970094 _ACL
+0x0a4 TokenType         : 1 ( TokenPrimary )
+0x0a8 ImpersonationLevel : 0 ( SecurityAnonymous )
+0x0ac TokenFlags        : 0xa50
+0x0b0 TokenInUse        : 0x1 ''
+0x0b4 IntegrityLevelIndex : 0xe
+0x0b8 MandatoryPolicy    : 3
+0x0bc LogonSession       : 0xbd0ebb0 _SEP_LOGON_SESSION_REFERENCES
+0x0c0 OriginatingLogonSession : _LUID
+0x0c8 SidHash           : _SID_AND_ATTRIBUTES_HASH
+0x150 RestrictedSidHash  : _SID_AND_ATTRIBUTES_HASH
+0x1d8 pSecurityAttributes : 0x919c35c8 _AUTHZBASEP_SECURITY_ATTRIBUTES_INFORMATION
+0x1dc VariablePart       : 0xd650ea7c
0: kd>

```

Рис. 2.5. Изучение структуры \_TOKEN

5. SID учетной записи пользователя-владельца маркера и групп, в которые он входит, хранятся по адресу в поле **UserAndGroups**. SID представляет собой уникальное значение переменной длины, используемое в операционных системах Windows для идентификации участника безопасности или группы безопасности. Чтобы его прочесть снова воспользуемся командой *dt* (см. рис. 2.6) Запишите результаты своих действий в отчет.

```

0: kd> dt _SID_AND_ATTRIBUTES 0xd650e9fc
nt!_SID_AND_ATTRIBUTES
+0x000 Sid              : 0xd650ea7c Void
+0x004 Attributes       : 0x10
0: kd>

```

Рис. 2.6. Изучение структуры SID\_AND\_ATTRIBUTES

6. В первом поле структуры \_SID\_AND\_ATTRIBUTES хранится адрес SID. Чтобы узнать какой SID расположен по данному адресу, можно воспользоваться следующей командой *!sid* (см. рис. 2.7). Запишите результаты своих действий в отчет.

```

0: kd> !sid 0xd650ea7c
SID is: S-1-5-21-2609833062-1600835590-1466217079-2004
0: kd>

```

Рис. 2.7. Получение SID

7. Сравните информацию, выводимую командой *!token*, с данными, полученными с помощью утилиты *Process Explorer*.

8. Подготовьте итоговый отчет с развернутыми выводами по заданию.

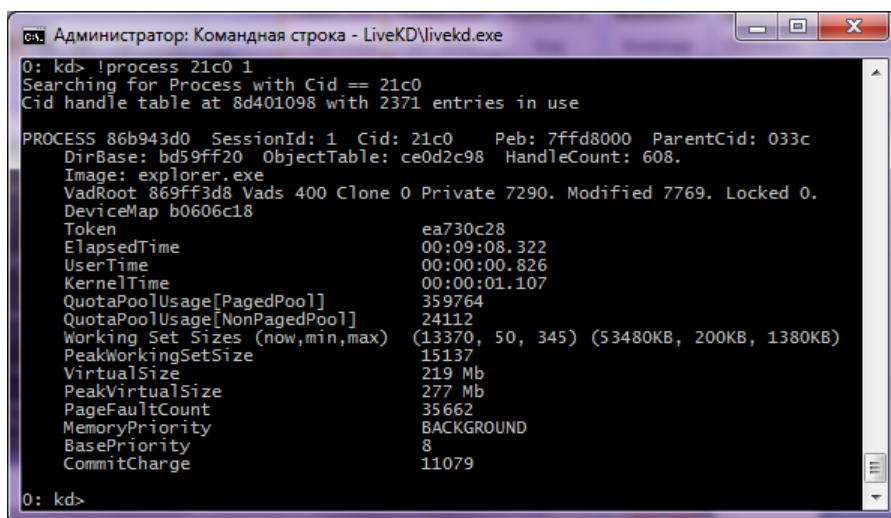
### Задание 2.3. Исследование дескриптора защиты (security descriptor)

Следуйте следующим указаниям при выполнении задания:

1. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту *LiveKd.exe*. В процессе запуска программы Вам могут быть заданы некоторые вопросы, касающиеся настроек запуска, отвечайте на них утвердительно.

2. Вызовите команду *!process 0 0*, для вывода краткого списка процессов системы, выберите процесс *explorer.exe* для дальнейшего изучения и запишите его идентификатор (тоже самое Вы можете выполнить с помощью утилиты *Process Explorer*). В нашем случае идентификатор процесса *explorer.exe* *21c0*. Запишите результаты выполнения данного пункта в отчет.

3. Выполните команду *!process <идентификатор процесса> 0*, в этом случае Вам отобразится подробная информация о процессе *explorer.exe*. Для дальнейшего изучения нам потребуется значение дескриптора объекта, для рассматриваемого примера это 86b943d0 (см. рис. 2.8). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.



```
0: kd> !process 21c0 1
Searching for Process with Cid == 21c0
Cid handle table at 8d401098 with 2371 entries in use

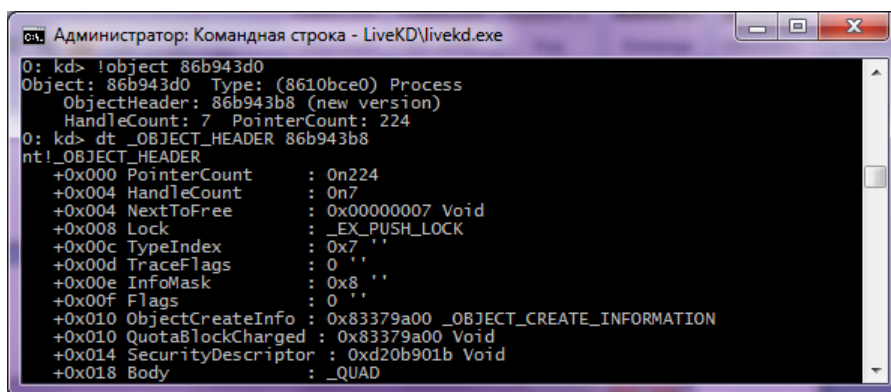
PROCESS 86b943d0 SessionId: 1 Cid: 21c0 Peb: 7ffd8000 ParentCid: 033c
DirBase: bd59ff20 ObjectTable: ce0d2c98 HandleCount: 608.
Image: explorer.exe
VadRoot 869ff3d8 Vads 400 Clone 0 Private 7290. Modified 7769. Locked 0.
DeviceMap b0606c18
Token ea730c28
ElapsedTime 00:09:08.322
UserTime 00:00:00.826
KernelTime 00:00:01.107
QuotaPoolUsage[PagedPool] 359764
QuotaPoolUsage[NonPagedPool] 24112
Working Set Sizes (now,min,max) (13370, 50, 345) (53480KB, 200KB, 1380KB)
PeakWorkingSetSize 15137
VirtualSize 219 Mb
PeakVirtualSize 277 Mb
PageFaultCount 35662
MemoryPriority BACKGROUND
BasePriority 8
CommitCharge 11079

0: kd>
```

Рис. 2.8. Получение подробной информации о процессе

4. Выполните команду `!process <идентификатор процесса> 0`, в этом случае Вам отобразится краткая информация о процессе `explorer.exe`. Для дальнейшего изучения нам потребуется значение дескриптора объекта, для рассматриваемого примера это `86b943d0`. Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.

5. С помощью команды `!object` определите адрес заголовка. Введите команду `dt _OBJECT_HEADER` и адрес поля заголовка объекта из вывода предыдущей команды для просмотра структуры данных заголовка объекта, включая значение указателя дескриптора защиты (см. рис. 2.9). Запишите результаты в отчет.



```
0: kd> !object 86b943d0
Object: 86b943d0  Type: (8610bce0) Process
ObjectHeader: 86b943b8 (new version)
HandleCount: 7  PointerCount: 224
0: kd> dt _OBJECT_HEADER 86b943b8
nt!_OBJECT_HEADER
+0x000 PointerCount      : 0n224
+0x004 HandleCount      : 0n7
+0x004 NextToFree       : 0x00000007 Void
+0x008 Lock             : _EX_PUSH_LOCK
+0x00c TypeIndex        : 0x7
+0x00d TraceFlags       : 0
+0x00e InfoMask         : 0x8
+0x00f Flags            : 0
+0x010 ObjectCreateInfo : 0x83379a00 _OBJECT_CREATE_INFORMATION
+0x010 QuotaBlockCharged : 0x83379a00 Void
+0x014 SecurityDescriptor : 0xd20b901b Void
+0x018 Body             : _QUAD
```

Рис. 2.9. Чтение структуры `dt _OBJECT_HEADER`

6. Выполните просмотр дескриптора безопасности объекта с помощью команды `!sd`. Используйте значение поля **SecurityDescriptor**, полученное на прошлом шаге с очищенными тремя или четырьмя последними битами (маски `-8` и `-10`) (см. рис. 2.10).

Уровень доступа к объекту определяется в списке DACL маской доступа. В маске отдельные биты отвечают за определенные виды доступа. Выделяют *стандартные права доступа* (Standard Access Rights), применимые к большинству объектов, и *специфичные для объектов права доступа* (Object-Specific Access Rights).



```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> !sd 0xd20b901b & -8
->Revision: 0x1
->Sbz1 : 0x0
->Control : 0x8814
          SE_DACL_PRESENT
          SE_SACL_PRESENT
          SE_SACL_AUTO_INHERITED
          SE_SELF_RELATIVE
->Owner : S-1-5-21-2609833062-1600835590-1466217079-2004
->Group : S-1-5-21-2609833062-1600835590-1466217079-513
->DACL :
->DACL : ->AceRevision: 0x2
->DACL : ->Sbz1 : 0x0
->DACL : ->AceSize : 0x68
->DACL : ->AceCount : 0x3
->DACL : ->Sbz2 : 0x0
->DACL : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->DACL : ->Ace[0]: ->AceFlags: 0x0
->DACL : ->Ace[0]: ->AceSize: 0x28
->DACL : ->Ace[0]: ->Mask : 0x001ffffff
->DACL : ->Ace[0]: ->SID: S-1-5-21-2609833062-1600835590-1466217079-2004
->DACL : ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->DACL : ->Ace[1]: ->AceFlags: 0x0
->DACL : ->Ace[1]: ->AceSize: 0x20
->DACL : ->Ace[1]: ->Mask : 0x00100001
->DACL : ->Ace[1]: ->SID: S-1-5-5-0-79882
->DACL : ->Ace[2]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->DACL : ->Ace[2]: ->AceFlags: 0x0
->DACL : ->Ace[2]: ->AceSize: 0x18
->DACL : ->Ace[2]: ->Mask : 0x00100000
->DACL : ->Ace[2]: ->SID: S-1-5-18
->SACL :
->SACL : ->AceRevision: 0x2
->SACL : ->Sbz1 : 0x0
->SACL : ->AceSize : 0x1c
->SACL : ->AceCount : 0x1
->SACL : ->Sbz2 : 0x0
->SACL : ->Ace[0]: ->AceType: SYSTEM_MANDATORY_LABEL_ACE_TYPE
->SACL : ->Ace[0]: ->AceFlags: 0x0
->SACL : ->Ace[0]: ->AceSize: 0x14
->SACL : ->Ace[0]: ->Mask : 0x00000003
->SACL : ->Ace[0]: ->SID: S-1-16-8192
0: kd>
```

Рис. 2.10. Просмотр дескриптора безопасности объекта с помощью

Для представленного примера дескриптор защиты содержит три элемента ACE типа «доступ разрешен», а также один элемент SACL, используемый для аудита доступа к объекту. Проанализируйте элементы ACE и их маски доступа к объекту для своего примера и сделайте в отчете их расшифровку.

7. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## РАБОТА 3. УПРАВЛЕНИЕ ФАЙЛОВОЙ СИСТЕМОЙ

Целью лабораторной работы №3 является исследование управления файловой системой с помощью Win32 API.

### Задание 3.1. Управление дисками, каталогами и файлами

Следуйте следующим указаниям при выполнении задания:

1. Создайте консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

- вывод списка дисков (функции Win32 API – **GetLogicalDrives**, **GetLogicalDriveStrings**);
- для одного из выбранных дисков вывод информации о диске и размер свободного пространства (функции Win32 API – **GetDriveType**, **GetVolumeInformation**, **GetDiskFreeSpace**);
- создание и удаление заданных каталогов (функции Win32 API – **CreateDirectory**, **RemoveDirectory**);
- создание файлов в новых каталогах (функция Win32 API – **CreateFile**)
- копирование и перемещение файлов между каталогами с возможностью выявления попытки работы с файлами, имеющими совпадающие имена (функции Win32 API – **CopyFile**, **MoveFile**, **MoveFileEx**);
- анализ и изменение атрибутов файлов (функции Win32 API – **GetFileAttributes**, **SetFileAttributes**, **GetFileInformationByHandle**, **GetFileTime**, **SetFileTime**).

2. Запустите приложение и проверьте его работоспособность на нескольких наборах вводимых данных. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Перезапустите приложение.



4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

5. Определите в **LiveKd** с помощью команды **!process** идентификатор процесса приложения. Запротоколируйте результаты в отчет.

6. В дальнейшем после выполнения каждого пункта меню делайте обновление снимка для **LiveKd** (нажимайте **Ctrl+Break** и затем 'y') и выполняйте с помощью команды **!handle** анализ используемых приложением объектов типа «файл». Протоколируйте результаты в отчет с комментариями изменений и раскрытием их взаимосвязи с выполненными инструкциями приложения.

7. С помощью утилиты **Handle** выполните закрытие открытого приложением объекта типа «файл», затем проверьте реакцию приложения. Запротоколируйте результаты в отчет.

8. Подготовьте итоговый отчет с развернутыми выводами по заданию.

### **Задание 3.2. Копирование файла с помощью операций перекрывающегося ввода-вывода**

Приложение должно копировать существующий файл в новый файл, «одновременно» выполняя *n* перекрывающихся операций ввода-вывода (механизм APC) блоками данных кратными размеру кластера.

Следуйте следующим указаниям при выполнении задания:

1. Создайте консольное приложение, которое выполняет:
  - открытие/создание файлов (функция Win32 API – **CreateFile**, обязательно использовать флаги FILE\_FLAG\_NO\_BUFFERING и FILE\_FLAG\_OVERLAPPED);
  - файловый ввод-вывод (функции Win32 API – **ReadFileEx**, **WriteFileEx**) блоками кратными размеру кластера;
  - ожидание срабатывания вызова функции завершения (функция Win32 API – **SleepEx**);

– измерение продолжительности выполнения операции копирования файла (функция Win32 API – **TimeGetTime**).

2. Запустите приложение и проверьте его работоспособность на копировании файлов разного размера для ситуации с перекрывающимся выполнением одной операции ввода и одной операции вывода (для сравнения файлов используйте консольную команду **FC**). Выполните эксперимент для разного размера копируемых блоков, постройте график зависимости скорости копирования от размера блока данных. Определите оптимальный размер блока данных, при котором скорость копирования наибольшая. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Произведите замеры времени выполнения приложения для разного числа перекрывающихся операций ввода и вывода (1, 2, 4, 8, 12, 16), не забывая проверять работоспособность приложения (консольная команда **FC**). По результатам измерений постройте график зависимости и определите число перекрывающихся операций ввода и вывода, при котором достигается наибольшая скорость копирования файла. Запротоколируйте результаты в отчет.

4. Перезапустите приложение.

5. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

6. Выполните анализ процесса приложения и его объектов (команды **!process, !handle**). Запротоколируйте результаты в отчет.

7. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## РАБОТА 4. УПРАВЛЕНИЕ ПАМЯТЬЮ

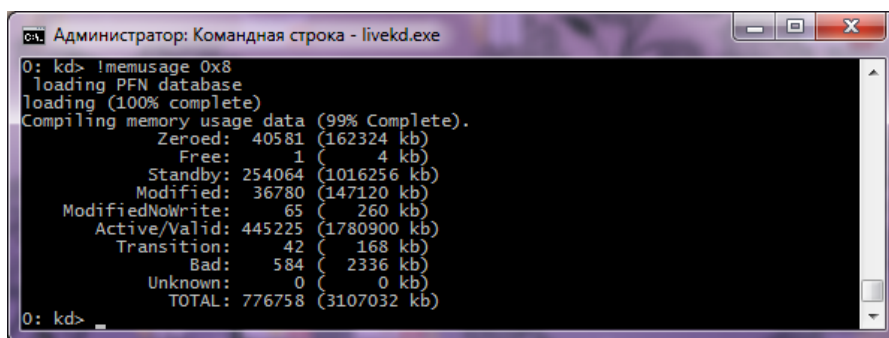
Целью лабораторной работы №4 является исследование механизмов управления виртуальной памятью Win32.

### Задание 4.1. Исследование процесса трансляции виртуальных адресов в 32-разрядной операционной системе Windows

Следуйте следующим указаниям при выполнении задания:

1. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**. В процессе запуска программы Вам могут быть заданы некоторые вопросы, касающиеся настроек запуска, отвечайте на них утвердительно.

2. Выполните команду **!memusage 0x8**, которая отображает размеры списков страниц виртуальной памяти, находящихся в различных состояниях (см. рис. 4.1). Ознакомьтесь с результатами выполнения и запишите их в отчет.



```
O: kd> !memusage 0x8
loading PFN database
loading (100% complete)
Compiling memory usage data (99% Complete).
Zeroed: 40581 (162324 kb)
Free: 1 (4 kb)
Standby: 254064 (1016256 kb)
Modified: 36780 (147120 kb)
ModifiedNoWrite: 65 (260 kb)
Active/Valid: 445225 (1780900 kb)
Transition: 42 (168 kb)
Bad: 584 (2336 kb)
Unknown: 0 (0 kb)
TOTAL: 776758 (3107032 kb)
O: kd>
```

Рис. 4.1. Просмотр размеров списков страниц виртуальной памяти

3. Выполните команду **!vm**, которая выводит базовые сведения об управлении памятью, доступные через соответствующие счетчики производительности. В том числе команда **!vm** выводит список процессов с их идентификаторами и объемом занимаемой памяти (см. рис. 4.2). Ознакомьтесь с результатами выполнения и запишите их в отчет. Выберите один из процессов для исследования, например, в дальнейшем мы будем изучать процесс **chrome.exe** с идентификатором 1918.

```
Администратор: Командная строка - livekd.exe
0: kd> !vm

*** Virtual Memory Usage ***
Physical Memory: 777342 ( 3109368 Kb)
Page File: \??\C:\pagefile.sys
Current: 7409084 Kb Free Space: 5635512 Kb
Minimum: 3109368 Kb Maximum: 9328104 Kb
Available Pages: 294318 ( 1177272 Kb)
ResAvail Pages: 635754 ( 2543016 Kb)
Locked IO Pages: 0 ( 0 Kb)
Free System PTEs: 106812 ( 427248 Kb)
Modified Pages: 15770 ( 63080 Kb)
Modified PF Pages: 15675 ( 62700 Kb)
NonPagedPool Usage: 30117 ( 120468 Kb)
NonPagedPool Max: 523068 ( 2092272 Kb)
PagedPool 0 Usage: 47600 ( 190400 Kb)
PagedPool 1 Usage: 7073 ( 28292 Kb)
PagedPool 2 Usage: 4245 ( 16980 Kb)
PagedPool 3 Usage: 4184 ( 16736 Kb)
PagedPool 4 Usage: 4225 ( 16900 Kb)
PagedPool Usage: 67327 ( 269308 Kb)
PagedPool Maximum: 523264 ( 2093056 Kb)

***** 6 pool allocations have failed *****

Session Commit: 23067 ( 92268 Kb)
Shared Commit: 117984 ( 471936 Kb)
Special Pool: 0 ( 0 Kb)
Shared Process: 4728 ( 18912 Kb)
PagedPool Commit: 67377 ( 269508 Kb)
Driver Commit: 7113 ( 28452 Kb)
Committed pages: 957888 ( 3831552 Kb)
Commit limit: 2629174 ( 10516696 Kb)

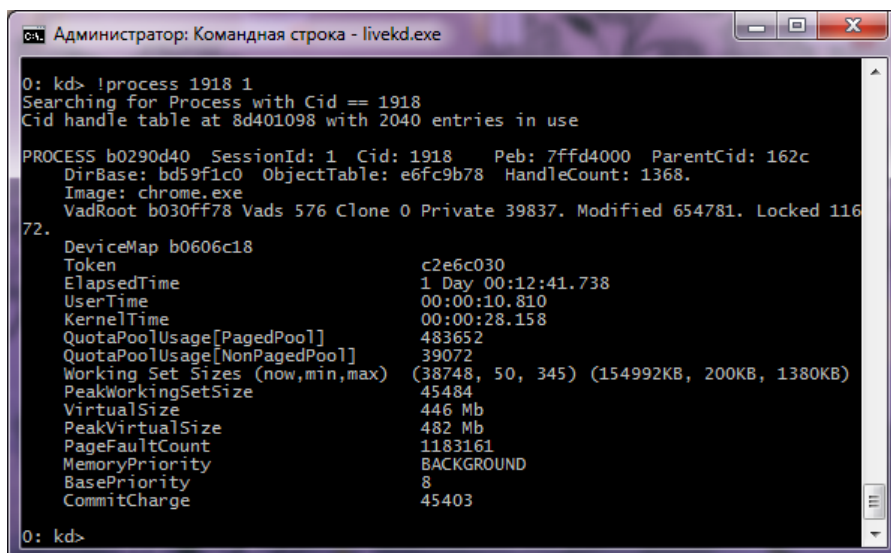
***** 1 commit requests have failed *****

Total Private: 686267 ( 2745068 Kb)
19ec avp.exe 98513 ( 394052 Kb)
275c WINWORD.EXE 71430 ( 285720 Kb)
2080 AcroRd32.exe 54842 ( 219368 Kb)
1918 chrome.exe 45403 ( 181612 Kb)
046c svchost.exe 41844 ( 167376 Kb)
```

Рис. 4.2. Вывод списка процессов с их идентификаторами и объемом занимаемой памяти

4. Чтобы просмотреть информацию о процессах, существует команда *!process 0 0*, которая выводит информацию о всех процессах. Ознакомьтесь с результатами выполнения, запишите их в отчет, выберите процесс для дальнейшего изучения его виртуального адресного пространства.

5. Выполните команду *!process <идентификатор процесса> 0*, в этом случае Вам отобразится краткая информация о выбранном процессе, например, адрес каталога страниц процесса **DirBase**, который понадобится для перехода в контекст процесса, также можно просмотреть адрес **PEB** процесса, имя файла процесса **Image**, адрес таблицы дескрипторов **ObjectTable**, размер таблицы дескрипторов **HandleCount**, адрес **VadRoot** корня дерева регионов виртуального адресного пространства процесса и т.д. Далее приведен пример вывода краткой информации о процессе *chrome.exe* с идентификатором *1918* (см. рис. 4.3). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.



```
0: kd> !process 1918 1
Searching for Process with Cid == 1918
Cid handle table at 8d401098 with 2040 entries in use

PROCESS b0290d40 SessionId: 1 Cid: 1918 Peb: 7ffd4000 ParentCid: 162c
DirBase: bd59f1c0 ObjectTable: e6fc9b78 HandleCount: 1368.
Image: chrome.exe
VadRoot b030ff78 Vads 576 Clone 0 Private 39837. Modified 654781. Locked 116
72.
DeviceMap b0606c18
Token c2e6c030
ElapsedTime 1 Day 00:12:41.738
UserTime 00:00:10.810
KernelTime 00:00:28.158
QuotaPoolUsage[PagedPool] 483652
QuotaPoolUsage[NonPagedPool] 39072
Working Set Sizes (now,min,max) (38748, 50, 345) (154992KB, 200KB, 1380KB)
PeakWorkingSetSize 45484
VirtualSize 446 Mb
PeakVirtualSize 482 Mb
PageFaultCount 1183161
MemoryPriority BACKGROUND
BasePriority 8
CommitCharge 45403
0: kd>
```

Рис. 4.3. Вывод информации о процессе

6. Далее проанализируем регионы виртуального адресного пространства выбранного процесса. Для этого необходимо вызвать команду `!vad <VadRoot>`. В результате выполнения этой команды мы получим краткую информацию о всех регионах виртуального адресного пространства выбранного процесса, в первом столбце выводится, а именно, виртуальный адрес описания региона (в скобках уровень иерархии в дереве), стартовый виртуальный адрес начала региона, адрес конца региона, статус региона и параметры защиты страниц. Список регионов может быть слишком большой, поэтому после запуска команды рекомендуется сразу приостановить вывод информации путем нажатия комбинации `Ctrl+Break`. Далее на рисунке 4.4 приведен пример вывода для процесса *chrome.exe* с идентификатором *1918*, у которого значение *VadRoot* было равно *b030aa78*. Обратите внимание на регион виртуальных адресов с адресом описания VAD *86825008*, в который спроецирован файл *\Program Files\Google\Chrome\chrome.exe*. Для получения подробной информации о выбранном регионе виртуального адресного пространства используем команду `!vad <VAD> 1` (см. рис. 4.5). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет. Далее попробуйте определить, какие страницы физической памяти поставлены в соответствие этому региону, и прочитайте из них информацию.

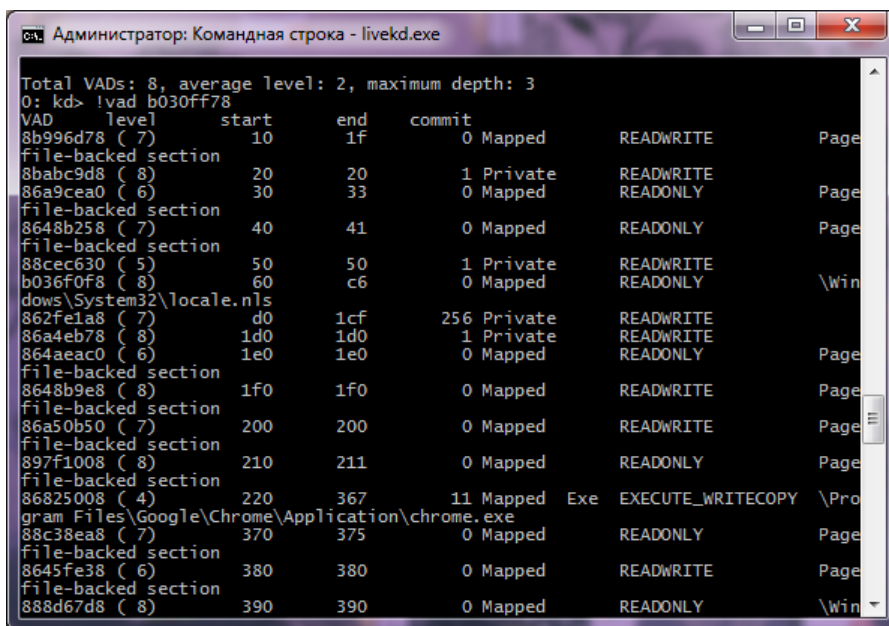


Рис. 4.4. Просмотр информации о регионах виртуального адресного пространства выбранного процесса

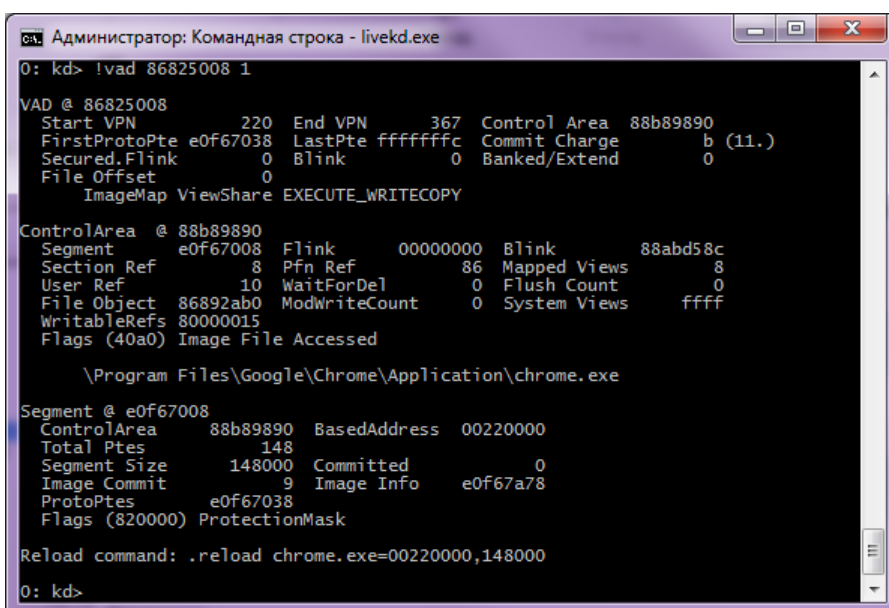
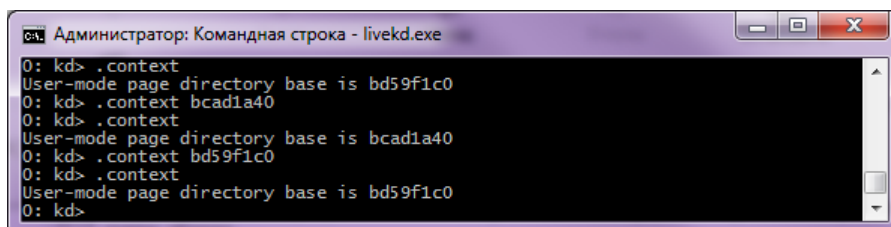


Рис. 4.5. Просмотр подробной информации о выбранном регионе виртуального адресного пространства

7. Чтобы получить доступ в виртуальное адресное пространство выбранного процесса необходимо, чтобы регистр CR3 указывал на каталог таблицы страниц этого процесса. Для решения этой проблемы мы воспользуемся командой перезагрузки контекста процессора в режиме отладчика *.context*. Вызовем эту команду без параметров, чтобы снова

проверить текущее значение базового адреса таблицы страниц, а затем вызовем команду *.context <DirBase >*. В нашем случае это *.context bd59f1c0*, затем снова проверим контекст процессора *.context* (см. рис. 4.7). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.



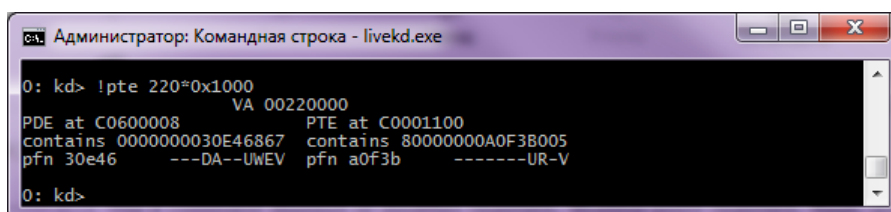
```

0: kd> .context
User-mode page directory base is bd59f1c0
0: kd> .context bcad1a40
0: kd> .context
User-mode page directory base is bcad1a40
0: kd> .context bd59f1c0
0: kd> .context
User-mode page directory base is bd59f1c0
0: kd>

```

Рис. 4.7. Переключение контекста процессора

8. После смены контекста можно смело выполнять доступ в виртуальную и физическую память процесса. Для начала необходимо преобразовать виртуальный адрес ячейки в физический адрес. Для этого необходимо обратиться в каталог страниц с помощью команды *!pte <виртуальный адрес>*, в нашем случае это *!pte <виртуальный адрес начала региона>*. Теперь мы имеем физический адрес стартовой страницы процесса, в которую операционной системой было выполнено проецирование файла *\Program Files\Google\Chrome\chrome.exe*. Для рассматриваемого примера это адрес *pfn a0f3b* (см. рис. 4.8). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.



```

0: kd> !pte 220*0x1000
VA 00220000
PDE at C0600008 PTE at C0001100
contains 0000000030E46867 contains 80000000A0F3B005
pfn 30e46 ---DA--UWV pfn a0f3b -----UR-V
0: kd>

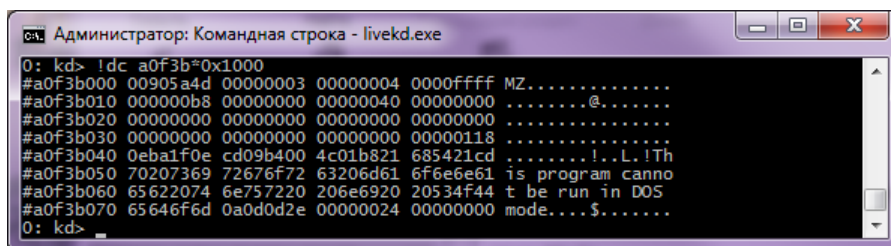
```

Рис. 4.8. Преобразование виртуального адреса ячейки в физический адрес

9. Теперь прочитаем ячейки оперативной памяти по адресу *pfn*, т.к. по этому адресу был спроецирован исполнимый файл, то в первых байтах должна располагаться специальная сигнатура **MZ**. Чтение физической памяти можно выполнить с помощью команды *!dc* (см. рис. 4.9). Повторите действия для выбранного процесса, ознакомьтесь с результатами и запишите их в отчет.



Удостоверьтесь, что в начале страницы действительно размещена сигнатура **MZ**.



```
0: kd> !dc a0f3b*0x1000
#a0f3b000 00905a4d 00000003 00000004 0000ffff MZ.....
#a0f3b010 000000b8 00000000 00000040 00000000 .....@.....
#a0f3b020 00000000 00000000 00000000 00000000 .....
#a0f3b030 00000000 00000000 00000000 00000118 .....
#a0f3b040 0eba1f0e cd09b400 4c01b821 685421cd .....!.L.!Th
#a0f3b050 70207369 72676f72 63206d61 6f6e6e61 is program canno
#a0f3b060 65622074 6e757220 206e6920 20534f44 t be run in DOS
#a0f3b070 65646f6d 0a0d0d2e 00000024 00000000 mode....$.
0: kd>
```

Рис. 4.9. Чтение сигнатуры исполняемого файла

10. Подготовьте отчет с развернутыми выводами по заданию.

#### Задание 4.2. Исследование виртуального адресного пространства процесса

Следуйте следующим указаниям при выполнении задания:

1. Создайте консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

- получение информации о вычислительной системе (функция Win32 API – **GetSystemInfo**);
- определение статуса виртуальной памяти (функция Win32 API – **GlobalMemoryStatus**);
- определение состояния конкретного участка памяти по заданному с клавиатуры адресу (функция Win32 API – **VirtualQuery**);
- резервирование региона в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – **VirtualAlloc**);
- резервирование региона и передача ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – **VirtualAlloc**);
- запись данных в ячейки памяти по заданным с клавиатуры адресам;
- установку защиты доступа для заданного (с клавиатуры) региона памяти и ее проверку (функция Win32 API – **VirtualProtect**);



– возврат физической памяти и освобождение региона адресного пространства, заданного с клавиатуры (функция Win32 API – **VirtualFree**).

2. Запустите приложение и проверьте его работоспособность на нескольких наборах вводимых данных. Запротоколируйте результаты в отчет. Дайте свои комментарии относительно выполнения функций Win32 API.

3. Перезапустите приложение.

4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

5. Определите для запущенного приложения в **LiveKd** с помощью команды **!process** идентификатор процесса, параметры виртуальной памяти, в том числе адрес **VadRoot** корня дерева регионов виртуального адресного пространства. Запротоколируйте результаты в отчет.

6. В дальнейшем после выполнения каждого пункта меню делайте обновление снимка для **LiveKd** (нажимайте **Ctrl+Break** и затем 'y') и выполняйте анализ виртуального адресного пространства процесса с помощью команды **!vad**. В случае выполнения в приложении записи в ячейки памяти данных выполните преобразование виртуальных адресов ячеек в физические (команда **!pte**) и затем вывод содержимого ячеек физической памяти на экран (команда **!dc**). Протоколируйте результаты в отчет с комментариями изменений и раскрытием их взаимосвязи с выполненными инструкциями приложения.

7. Подготовьте итоговый отчет с развернутыми выводами по заданию.

#### **Задание 4.3. Использование проецируемых файлов для обмена данными между процессами**

Следуйте следующим указаниям при выполнении задания:

1. Создайте два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которые выполняют:

- приложение-писатель создает проецируемый файл (функции Win32 API – **CreateFile**, **CreateFileMapping**), проецирует фрагмент файла в память (функции Win32 API – **MapViewOfFile**, **UnmapViewOfFile**), осуществляет ввод данных с клавиатуры и их запись в спроецированный файл;
- приложение-читатель открывает проецируемый файл (функция Win32 API – **OpenFileMapping**), проецирует фрагмент файла в память (функции Win32 API – **MapViewOfFile**, **UnmapViewOfFile**), считывает содержимое из спроецированного файла и отображает на экран.

2. Запустите приложения и проверьте обмен данных между процессами, удостоверьтесь в надлежащем выполнении задания. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Перезапустите разработанные приложения.

4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

5. Определите в **LiveKd** с помощью команды **!process** идентификаторы процессов приложений. Запротоколируйте результаты в отчет.

6. В дальнейшем после выполнения каждого пункта меню делайте обновление снимка для **LiveKd** (нажимайте **Ctrl+Break** и затем 'y'), получайте информацию с помощью команды **!handle** об объектах обоих процессов и выполняйте анализ виртуальных адресных пространств процессов с помощью команды **!vad**. После выполнения в приложении записи в выделенные ячейки памяти данных выполните для обоих процессов преобразование виртуальных адресов ячеек в физические (команда **!pte**) и вывод содержимого ячеек физической памяти на экран (команда **!dc**). Протоколируйте результаты в отчет с комментариями изменений и раскрытием их взаимосвязи с выполненными инструкциями приложений.

7. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## РАБОТА 5. ПРОЦЕССЫ И ПОТОКИ

Целью лабораторной работы №5 является исследование механизмов создания и управления процессами и потоками в ОС Windows.

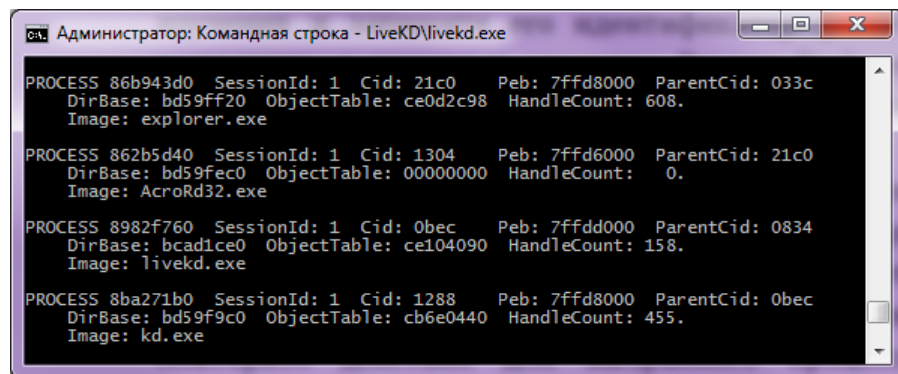
### Задание 5.1. Исследование структур данных процессов и потоков

Следуйте следующим указаниям при выполнении задания:

1. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

2. С помощью команды **!process 0 0** получите информацию обо всех процессах системы, в том числе их идентификаторы и адреса структур EPROCESS. Команда **!process** отображает информацию обо всех или нескольких процессах. Первый ноль в параметрах команды означает, что нужно выводить информацию обо всех процессах. Если на месте первого параметра указать ID процесса или адрес в памяти его структуры EPROCESS, будет выводиться информация только о данном процессе. Второй ноль в параметрах команды **!process** определяет количество информации о процессе: 0 – минимум информации, 7 – максимум информации.

В представленном на рисунке 5.1 примере ID процесса **explorer.exe** равен 0x21c0, а адрес структуры EPROCESS = 86b943d0.



```
Администратор: Командная строка - LiveKD\livekd.exe

PROCESS 86b943d0 SessionId: 1 Cid: 21c0 Peb: 7ffd8000 ParentCid: 033c
DirBase: bd59ff20 ObjectTable: ce0d2c98 HandleCount: 608.
Image: explorer.exe

PROCESS 862b5d40 SessionId: 1 Cid: 1304 Peb: 7ffd6000 ParentCid: 21c0
DirBase: bd59fec0 ObjectTable: 00000000 HandleCount: 0.
Image: AcroRd32.exe

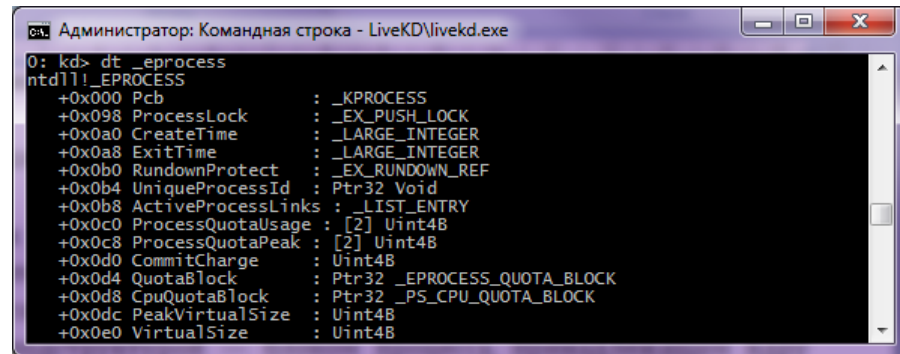
PROCESS 8982f760 SessionId: 1 Cid: 0bec Peb: 7ffdd000 ParentCid: 0834
DirBase: bcad1ce0 ObjectTable: ce104090 HandleCount: 158.
Image: livekd.exe

PROCESS 8ba271b0 SessionId: 1 Cid: 1288 Peb: 7ffd8000 ParentCid: 0bec
DirBase: bd59f9c0 ObjectTable: cb6e0440 HandleCount: 455.
Image: kd.exe
```

Рис. 5.1. Получение информации обо всех процессах системы

Список полей, составляющих блок EPROCESS, и их смещения в шестнадцатеричной форме, можно увидеть с помощью команды **dt \_eprocess** отладчика ядра (см. рис. 5.2). Слева в окне вывода указывается

шестнадцатеричное смещение в байтах для поля относительно начала расположения структуры в памяти. Заметьте, что первое поле (Pcb) на самом деле является подструктурой – блоком процесса, принадлежащим ядру (KPROCESS). Именно здесь хранится информация, используемая при планировании.



```
0: kd> dt _eprocess
ntdll!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x098 ProcessLock : _EX_PUSH_LOCK
+0x0a0 CreateTime : _LARGE_INTEGER
+0x0a8 ExitTime : _LARGE_INTEGER
+0x0b0 RundownProtect : _EX_RUNDOWN_REF
+0x0b4 UniqueProcessId : Ptr32 Void
+0x0b8 ActiveProcessLinks : _LIST_ENTRY
+0x0c0 ProcessQuotaUsage : [2] UInt4B
+0x0c8 ProcessQuotaPeak : [2] UInt4B
+0x0d0 CommitCharge : UInt4B
+0x0d4 QuotaBlock : Ptr32 _EPROCESS_QUOTA_BLOCK
+0x0d8 CpuQuotaBlock : Ptr32 _PS_CPU_QUOTA_BLOCK
+0x0dc PeakVirtualSize : UInt4B
+0x0e0 VirtualSize : UInt4B
```

Рис. 5.2. Просмотр полей структуры EPROCESS

Выведите список процессов системы, выберите процесс для дальнейшего изучения. Запротоколируйте результаты в отчет с комментариями.

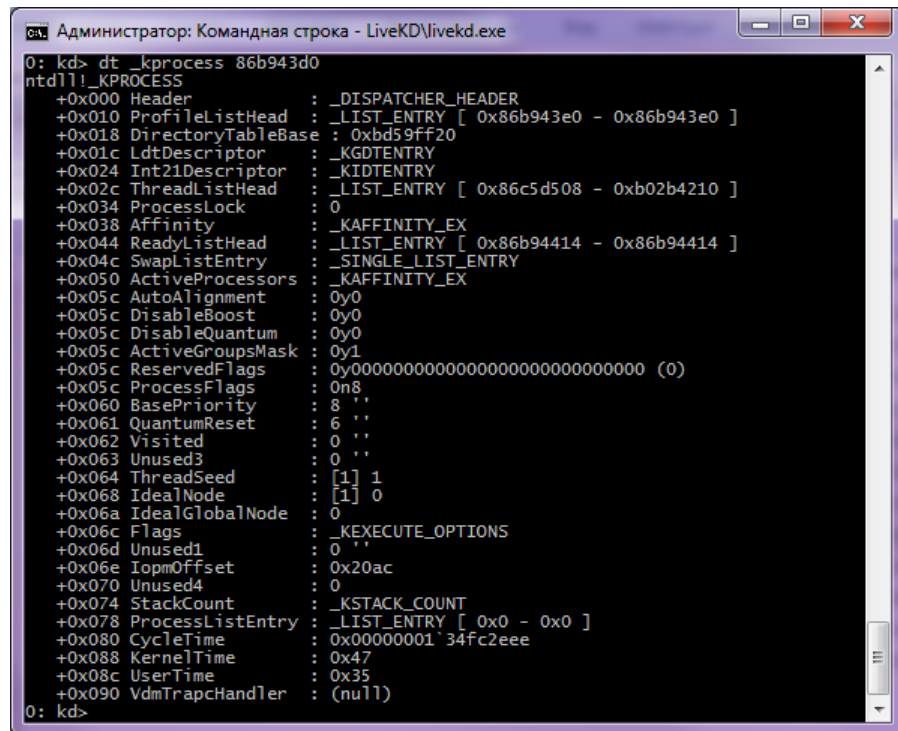
3. Чтобы отобразить значения полей структуры `_EPROCESS` для какого-либо конкретного процесса, необходимо воспользоваться командой `dt _eprocess <Pcb>`. Например, для вывода значения полей структуры EPROCESS для процесса **explorer.exe** необходимо ввести команду: `dt eprocess 86b943d0` (см. рис. 5.3).

Обратите внимание на идентификатор процесса – поле **UniqueProcessId** (его значение должно совпадать с полученным ранее идентификатором), и на файл образа процесса – поле **ImageFileName** (**explorer.exe**). Класс приоритета процесса хранится в поле **PriorityClass**. Базовый приоритет процесса хранится в поле **BasePriority**. Значение базового приоритета должно соответствовать классу приоритета процесса.

```
0: kd> dt ntl_eprocess 86b943d0
+0x000 Pcb : _KPROCESS
+0x098 ProcessLock : _EX_PUSH_LOCK
+0x0a0 CreateTime : _LARGE_INTEGER 0x01ce5c49`0920787c
+0x0a8 ExitTime : _LARGE_INTEGER 0x0
+0x0b0 RundownProtect : _EX_RUNDOWN_REF
+0x0b4 UniqueProcessId : 0x000021c0 Void
+0x0b8 ActiveProcessLinks : _LIST_ENTRY [ 0x862b5df8 - 0xd917ac80 ]
+0x0c0 ProcessQuotaUsage : [2] 0x5e30
+0x0c8 ProcessQuotaPeak : [2] 0x746c
+0x0d0 CommitCharge : 0x2b47
+0x0d4 QuotaBlock : 0x887da640 _EPROCESS_QUOTA_BLOCK
+0x0d8 CpuQuotaBlock : (null)
+0x0dc PeakVirtualSize : 0x1154c000
+0x0e0 VirtualSize : 0xdb4c000
+0x0e4 SessionProcessLinks : _LIST_ENTRY [ 0x862b5e24 - 0xd917acac ]
+0x0ec DebugPort : (null)
+0x0f0 ExceptionPortData : 0x886099f8 Void
+0x0f0 ExceptionPortValue : 0x886099f8
+0x0f0 ExceptionPortState : 0y000
+0x0f4 ObjectTable : 0xce0d2c98 _HANDLE_TABLE
+0x0f8 Token : _EX_FAST_REF
+0x0fc WorkingSetPage : 0x2d6ad
+0x100 AddressCreationLock : _EX_PUSH_LOCK
+0x104 RotateInProgress : (null)
+0x108 ForkInProgress : (null)
+0x10c HardwareTrigger : 0
+0x110 PhysicalVadRoot : (null)
+0x114 CloneRoot : (null)
+0x118 NumberOfPrivatePages : 0x1c7a
+0x11c NumberOfLockedPages : 0
+0x120 Win32Process : 0xfb393b30 Void
+0x124 Job : (null)
+0x128 SectionObject : 0xcb789350 Void
+0x12c SectionBaseAddress : 0x00d60000 Void
+0x130 Cookie : 0x437916c7
+0x134 Spare8 : 0
+0x138 WorkingSetWatch : (null)
+0x13c Win32WindowStation : 0x00000030 Void
+0x140 InheritedFromUniqueProcessId : 0x0000033c Void
+0x144 LdtInformation : (null)
+0x148 VdmObjects : (null)
+0x14c ConsoleHostProcess : 0
+0x150 DeviceMap : 0xb0606c18 Void
+0x154 EtwDataSource : (null)
+0x158 FreeTebHint : 0x7ffdc000 Void
+0x160 PageDirectoryPte : _HARDWARE_PTE
+0x160 Filler : 0
+0x168 Session : 0x90299000 Void
+0x16c ImageFileName : [15] "explorer.exe"
+0x17b PriorityClass : 0x2 ''
+0x17c JobLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
+0x184 LockedPagesList : (null)
+0x188 ThreadListHead : _LIST_ENTRY [ 0x86c5d590 - 0xb02b4298 ]
```

Рис. 5.3. Чтение структуры EPROCESS для выбранного процесса

Структура KPROCESS содержится в поле **Pcb** структуры EPROCESS, причем это поле находится в самом начале структуры (смещение равно нулю). Поэтому можно отобразить структуру KPROCESS с того же адреса, по которому располагается EPROCESS: *dt \_kprocess 86b943d0* (см. рис. 5.4).



```
0: kd> dt _kprocess 86b943d0
ntdll!_KPROCESS
+0x000 Header          : _DISPATCHER_HEADER
+0x010 ProfileListHead : _LIST_ENTRY [ 0x86b943e0 - 0x86b943e0 ]
+0x018 DirectoryTableBase : 0xbd59ff20
+0x01c LdtDescriptor    : _KGDTENTRY
+0x024 Int21Descriptor  : _KIDTENTRY
+0x02c ThreadListHead  : _LIST_ENTRY [ 0x86c5d508 - 0xb02b4210 ]
+0x034 ProcessLock     : 0
+0x038 Affinity        : _KAFFINITY_EX
+0x044 ReadyListHead   : _LIST_ENTRY [ 0x86b94414 - 0x86b94414 ]
+0x04c SwapListEntry   : _SINGLE_LIST_ENTRY
+0x050 ActiveProcessors : _KAFFINITY_EX
+0x05c AutoAlignment   : 0y0
+0x05c DisableBoost    : 0y0
+0x05c DisableQuantum  : 0y0
+0x05c ActiveGroupsMask : 0y1
+0x05c ReservedFlags   : 0y00000000000000000000000000000000 (0)
+0x05c ProcessFlags    : 0n8
+0x060 BasePriority     : 8
+0x061 QuantumReset    : 6
+0x062 Visited         : 0
+0x063 Unused3        : 0
+0x064 ThreadSeed      : [1] 1
+0x068 IdealNode       : [1] 0
+0x06a IdealGlobalNode : 0
+0x06c Flags           : _KEXECUTE_OPTIONS
+0x06d Unused1         : 0
+0x06e IoPmOffset      : 0x20ac
+0x070 Unused4         : 0
+0x074 StackCount      : _KSTACK_COUNT
+0x078 ProcessListEntry : _LIST_ENTRY [ 0x0 - 0x0 ]
+0x080 CycleTime       : 0x00000001'34fc2eee
+0x088 KernelTime      : 0x47
+0x08c UserTime        : 0x35
+0x090 VdmTrapHandler  : (null)
0: kd>
```

Рис. 5.4. Чтение структуры KPROCESS для выбранного процесса

В этой структуре KPROCESS в поле **QuantumReset** хранится величина кванта для потоков процесса. Значение поля **QuantumReset** равно 6 единицам, что составляет 12 интервалов системного таймера.

Получите структуры EPROCESS, KPROCESS для выбранного процесса. Запротоколируйте результаты в отчет с комментариями.

4. Для получения информации о потоках процесса. Чтобы вывести информацию о потоках процесса, наберите команду: **!process <PID> 4**. Для каждого потока указывается адрес его структуры ETHREAD, идентификатор потока и текущее состояние потока.

Для получения информации о потоке используйте команду **!thread address**, где **address** – адрес структуры ETHREAD потока.

Для просмотра значений полей структуры ETHREAD для конкретного потока используйте команду: **dt \_ethread address**.

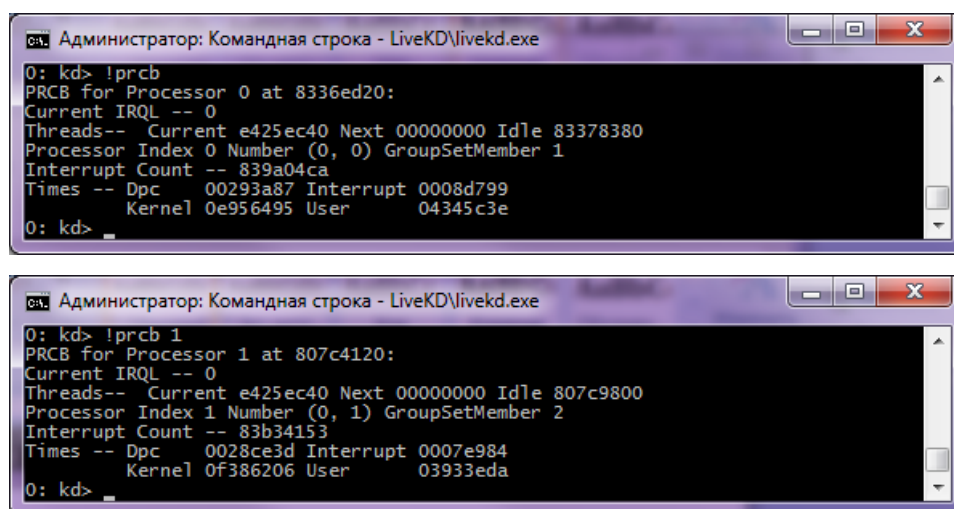
Получите список потоков выбранного процесса, выберите несколько потоков процесса и получите для них структуры ETHREAD. Запротоколируйте результаты в отчет с комментариями.

5. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## Задание 5.2. Исследование регистра контроля процессора и очередь потоков готовых для выполнения

Следуйте следующим указаниям при выполнении задания:

1. Каждый процессор обладает так называемым PCR – «регистром контроля процессора» (processor control register), который хранит информацию о таких вещах, как уровень IRQL, GDT, IDT и т.д. Для получения регистра контроля процессора следует воспользоваться командой *!prcb* <номер процессора> (см. рис. 5.5). В результате выполнения команды Вы получите информацию о процессоре, уровень IRQL, адреса структур ETHREAD для текущего потока, следующего потока и потока простоя, а также адрес структуры KPRCB с расширенной информацией о регистре контроля процессора.



```
0: kd> !prcb
PRCB for Processor 0 at 8336ed20:
Current IRQL -- 0
Threads-- Current e425ec40 Next 00000000 Idle 83378380
Processor Index 0 Number (0, 0) GroupSetMember 1
Interrupt Count -- 839a04ca
Times -- Dpc 00293a87 Interrupt 0008d799
Kernel 0e956495 User 04345c3e
0: kd>

0: kd> !prcb 1
PRCB for Processor 1 at 807c4120:
Current IRQL -- 0
Threads-- Current e425ec40 Next 00000000 Idle 807c9800
Processor Index 1 Number (0, 1) GroupSetMember 2
Interrupt Count -- 83b34153
Times -- Dpc 0028ce3d Interrupt 0007e984
Kernel 0f386206 User 03933eda
0: kd>
```

Рис. 5.5. Чтение регистра контроля процессора

Выполните команду *!prcb* для всех процессоров Вашей вычислительной системы. Запротоколируйте результаты в отчет.

2. Скачайте и запустите утилиту **CPUSTRES** (<http://live.sysinternals.com/WindowsInternals/>).

Выберите значения полей утилиты так, как показано на рисунке 5.6. Важно обеспечить загрузку системы несколькими потоками, желательно с разными приоритетами.



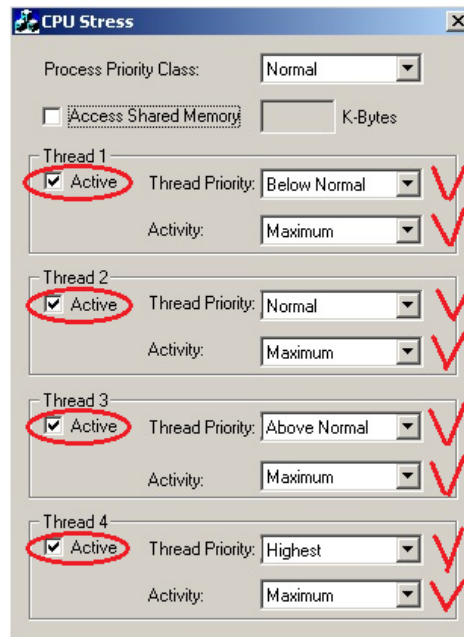


Рис. 5.6. Настройка утилиты CPUSTRES

3. Для получения структуры KPRCB (Kernel Processor Register Control Block) для выбранного процессора вызовите команду *dt \_kprcb <адрес PRCB>*. Например, для процессора «0» из примера предыдущего пункта *dt \_kprcb 8336ed20* (см. рис. 5.7).

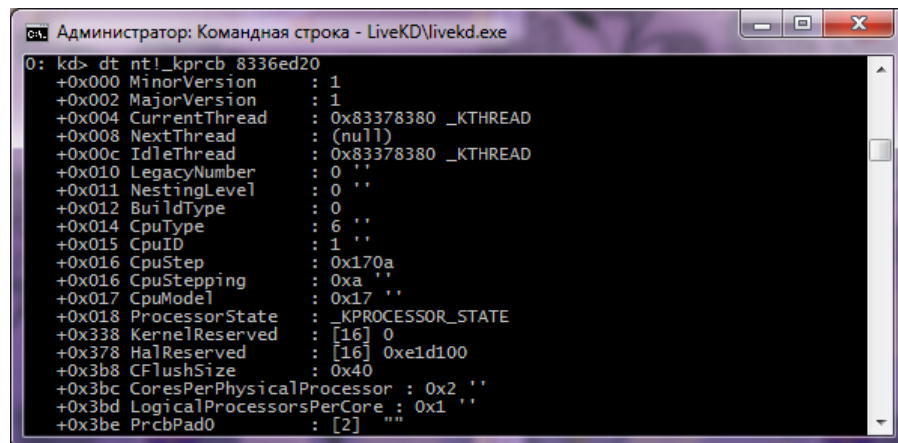


Рис. 5.7. Получение структуры KPRCB для выбранного процессора

Найдите поля **ReadySummary** и **DispatcherReadyListHead**. Поле **ReadySummary** в битовом формате показывает приоритеты, для которых имеются готовые к выполнению потоки. Это поле используется для ускорения поиска очереди потоков с максимальным приоритетом: система не просматривает все очереди для каждого приоритета, а сначала обращается к полю **ReadySummary**, чтобы найти готовый поток с максимальным



приоритетом. В данном примере для процессора «0» это поток с приоритетом 8, а для процессора «1» – поток с приоритетом 10 (см. рис. 5.8).

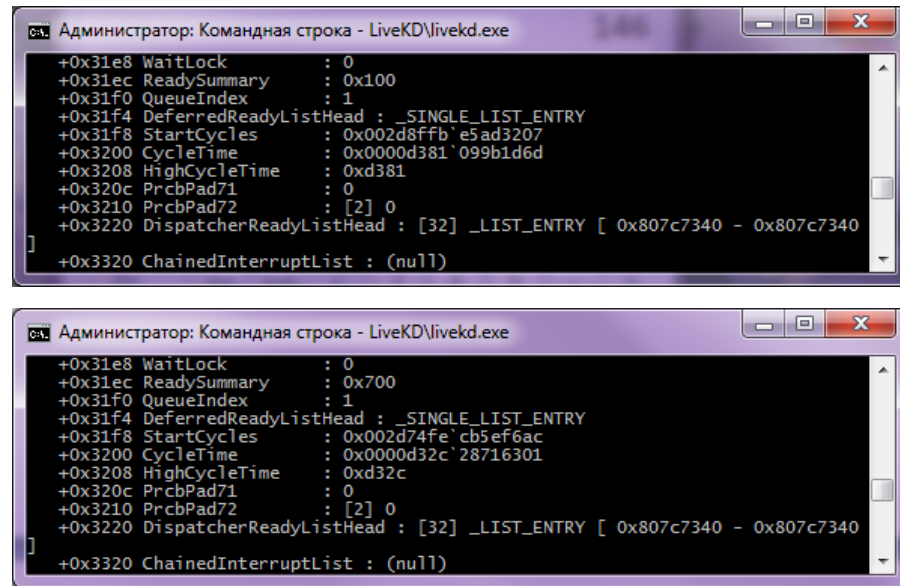


Рис. 5.8. Анализ поля ReadySummary для выбранного процессора

Поле **DispatcherReadyListHead** указывает на очереди готовых потоков. Данное поле представляет собой массив элементов типа **LIST\_ENTRY**. Размерность массива совпадает с количеством приоритетов в системе – 32 элемента.

Определите параметры очередей готовых потоков для всех процессоров Вашей вычислительной системы. Запротоколируйте результаты в отчет.

6. Подготовьте итоговый отчет с развернутыми выводами по заданию.

### Задание 5.3. Реализация многопоточного приложения с использованием функций Win32 API

Следуйте следующим указаниям при выполнении задания:

1. Создайте приложение, которое вычисляет число  $\pi$  с точностью N знаков после запятой по следующей формуле

$$\pi = \left( \frac{4}{1+x_0^2} + \frac{4}{1+x_1^2} + \dots + \frac{4}{1+x_{N-1}^2} \right) \times \frac{1}{N}, \text{ где } x_i = (i+0.5) \times \frac{1}{N}, i = \overline{0, N-1}, \text{ где}$$

N=10000000.

- Используйте распределение итераций блоками (размер блока =  $10 * N_{\text{студбилета}}$ ) по потокам. Сначала каждый поток по очереди получает свой блок итераций, затем тот поток, который заканчивает выполнение своего блока, получает следующий свободный блок итераций. Освободившиеся потоки получают новые блоки итераций до тех пор, пока все блоки не будут исчерпаны.
- Создание потоков выполняйте с помощью функции Win32 API **CreateThread**.
- Для реализации механизма распределения блоков итераций необходимо сразу в начале программы создать необходимое количество потоков в приостановленном состоянии, для освобождения потока из приостановленного состояния используйте функцию Win32 API **ResumeThread**.
- По окончании обработки текущего блока итераций поток не должен завершаться, а должен быть приостановлен с помощью функции Win32 API **SuspendThread**. Затем потоку должна быть предоставлена следующий свободный блок итераций, и поток должен быть освобожден (**ResumeThread**).
- Для хранения потоками промежуточных результатов расчета числа  $\pi$  используйте механизм TLS (нечетные номера студенческого билета – статический TLS, четные номера билета – динамический TLS).

2. Произведите замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам измерений постройте график и определите число потоков, при котором достигается наибольшая скорость выполнения. Запротоколируйте результаты в отчет.

3. Перезапустите приложение с числом потоков более 4-х.

4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

5. С помощью команды *!process* определите идентификатор процесса приложения и перечень потоков, получите с помощью *dt \_eprocess*, *dt \_kthread* сведения о процессе и его потоках. С помощью *!prcb*, *dt \_kprcb*, *dt \_kthread* определите сведения о текущем потоке, следующем потоке и потоке простоя. Запротоколируйте результаты в отчет.

6. Подготовьте итоговый отчет с развернутыми выводами по заданию.

#### **Задание 5.4. Реализация многопоточного приложения с использованием технологии OpenMP**

Следуйте следующим указаниям при выполнении задания:

1. Создайте приложение, которое вычисляет число пи с точностью N знаков после запятой по формуле из задания 5.4.

- Распределите работу по потокам с помощью OpenMP-директивы **for**.
- Используйте динамическое планирование блоками итераций (размер блока =  $10 * N_{\text{студбилета}}$ ).

2. Произведите замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам измерений постройте график и определите число потоков, при котором достигается наибольшая скорость выполнения. Запротоколируйте результаты в отчет, сравните с результатами прошлой работы.

3. Запустите приложение с числом потоков более 4-х.

4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

5. С помощью команды *!process* определите идентификатор процесса приложения и перечень потоков, получите с помощью *dt \_eprocess*, *dt \_kthread* сведения о процессе и его потоках. С помощью *!prcb*, *dt \_kprcb*, *dt \_kthread* определите сведения о текущем потоке, следующем потоке и потоке простоя. Запротоколируйте результаты в отчет, сравните с прошлым заданием.

6. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## РАБОТА 6. МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ

Целью лабораторной работы №6 является исследование инструментов и механизмов взаимодействия процессов в Windows.

### Задание 6.1. Реализация решения задачи о читателях-писателях.

Следуйте следующим указаниям при выполнении задания:

1. Выполнить решение задачи о читателях-писателях, для чего необходимо разработать консольные приложения «Читатель» и «Писатель»:

- одновременно запущенные экземпляры процессов-читателей и процессов-писателей должны совместно работать с буферной памятью в виде проецируемого файла: размер страницы буферной памяти равен размеру физической страницы оперативной памяти; число страниц буферной памяти равно сумме цифр в номере студенческого билета без учета первой цифры.
- страницы буферной памяти должны быть заблокированы в оперативной памяти (функция **VirtualLock**);
- длительность выполнения процессами операций «чтения» и «записи» задается случайным образом в диапазоне от 0,5 до 1,5 сек.;
- для синхронизации работы процессов необходимо использовать объекты синхронизации типа «семафор» и «мьютекс»;
- процессы-читатели и процессы-писатели ведут свои журнальные файлы, в которые регистрируют переходы из одного «состояния» в другое (начало ожидания, запись или чтение, переход к освобождению) с указанием кода времени (функция **TimeGetTime**). Для состояний «запись» и «чтение» необходимо также запротоколировать номер рабочей страницы.

2. Запустите приложения читателей и писателей, суммарное количество одновременно работающих читателей и писателей должно быть не менее числа страниц буферной памяти. Проверьте функционирование приложений, проанализируйте журнальные файлы процессов, постройте сводные графики

смены «состояний» для не менее 5 процессов-читателей и 5 процессов-писателей, дайте свои комментарии относительно переходов процессов из одного состояния в другое. Постройте графики занятости страниц буферной памяти (проецируемого файла) во времени, дайте свои комментарии.

3. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

4. Определите в **LiveKd** с помощью команды **!process** идентификаторы приложений читателей и писателей, получите информацию с помощью команды **!handle** об объектах процессов. Запротоколируйте результаты в отчет.

5. Подготовьте итоговый отчет с развернутыми выводами по заданию.

**Задание 6.2.** Использование именованных каналов для реализации сетевого межпроцессного взаимодействия.

Следуйте следующим указаниям при выполнении задания:

1. Создайте два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которые выполняют:

- приложение-сервер создает именованный канал (функция Win32 API – **CreateNamedPipe**), выполняет установление и отключение соединения (функции Win32 API – **ConnectNamedPipe**, **DisconnectNamedPipe**), создает объект «событие» (функция Win32 API – **CreateEvent**) осуществляет ввод данных с клавиатуры и их асинхронную запись в именованный канал (функция Win32 API – **WriteFile**), выполняет ожидание завершения операции ввода-вывода (функция Win32 API – **WaitForSingleObject**);
- приложение-клиент подключается к именованному каналу (функция Win32 API – **CreateFile**), в асинхронном режиме считывает содержимое из

именованного канала файла (функция Win32 API – **ReadFileEx**) и отображает на экран.

2. Запустите приложения и проверьте обмен данных между процессами. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Перезапустите разработанные приложения.

4. Запустите командную строку от имени администратора, перейдите в каталог **c:\Tools\LiveKD** и запустите утилиту **LiveKd.exe**.

5. Определите в **LiveKd** с помощью команды **!process** идентификаторы приложений. Запротоколируйте результаты в отчет.

6. Сделайте несколько снимков системы для **LiveKd** (нажимайте **Ctrl+Break** и затем 'y'), получайте информацию с помощью команды **!handle** об объектах процессов. Запротоколируйте результаты в отчет.

7. Подготовьте итоговый отчет с развернутыми выводами по заданию.

## Содержание

Лабораторная работа № 1. Исследование объектов Windows.....	3
Лабораторная работа № 2. Исследование структур данных обеспечения безопасности в WINDOWS.....	16
Лабораторная работа № 3. Управление файловой системой.....	24
Лабораторная работа № 4. Управление памятью.....	27
Лабораторная работа № 5. Процессы и потоки.....	35
Лабораторная работа № 6. Межпроцессное взаимодействие.....	44

В авторской редакции.