

## Работа 5. Процессы и потоки

**Цель работы:** исследовать механизмы создания и управления процессами и потоками в ОС Windows.

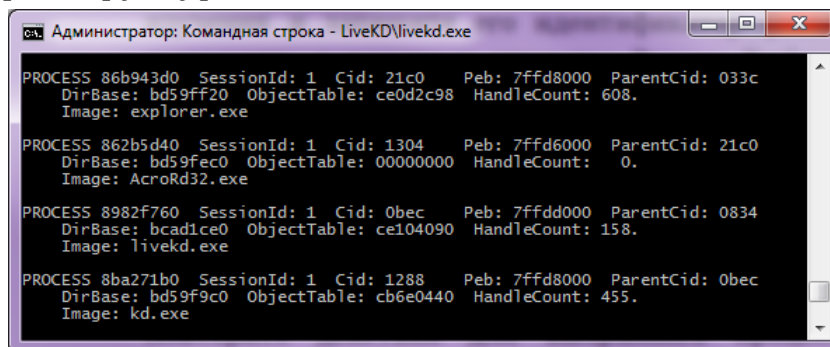
**Задание 5.1.** Исследовать структуры данных процессов и потоков.

**Указания к выполнению.**

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 ниже примере ID процесса **explorer.exe** равен 0x21c0, а адрес структуры EPROCESS = 86b943d0.



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

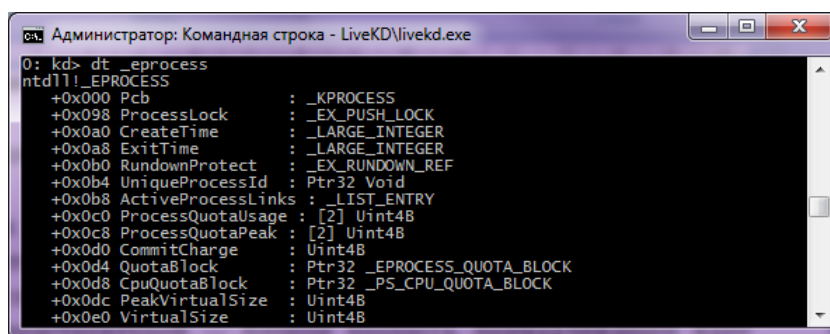
PROCESS 862b5d40 SessionId: 1 Cid: 1304 Peb: 7ffd6000 ParentCid: 21c0
DirBase: bd59fec0 ObjectTable: 00000000 HandleCount: 0.
Image: AcroRd32.exe

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
```

Список полей, составляющих блок EPROCESS, и их смещения в шестнадцатеричной форме, можно увидеть с помощью команды *dt \_eprocess* отладчика ядра. 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

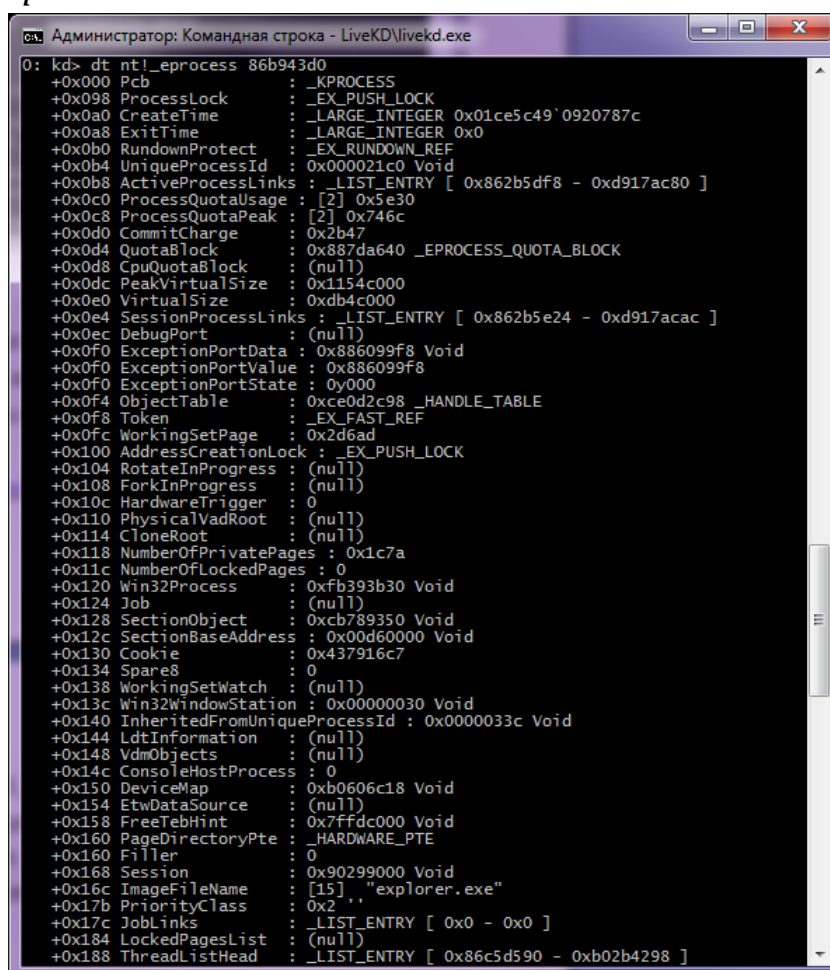
ядру (KPROCESS). Именно здесь хранится информация, используемая при планировании.



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

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`.



```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> dt nt!_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 ]
```

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.

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*.

```

0: kd> dt _kprocess 86b943d0
ntdll!_KPROCESS
+0x000 Header : _DISPATCHER_HEADER
+0x010 ProfileListHead : _LIST_ENTRY [ 0x86b943e0 - 0x86b943e0 ]
+0x018 DirectoryTableBase : 0xbd59ff20
+0x01c LdtDescriptor : _KGDENTRY
+0x024 Int21Descriptor : _KIDENTRY
+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>

```

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*.

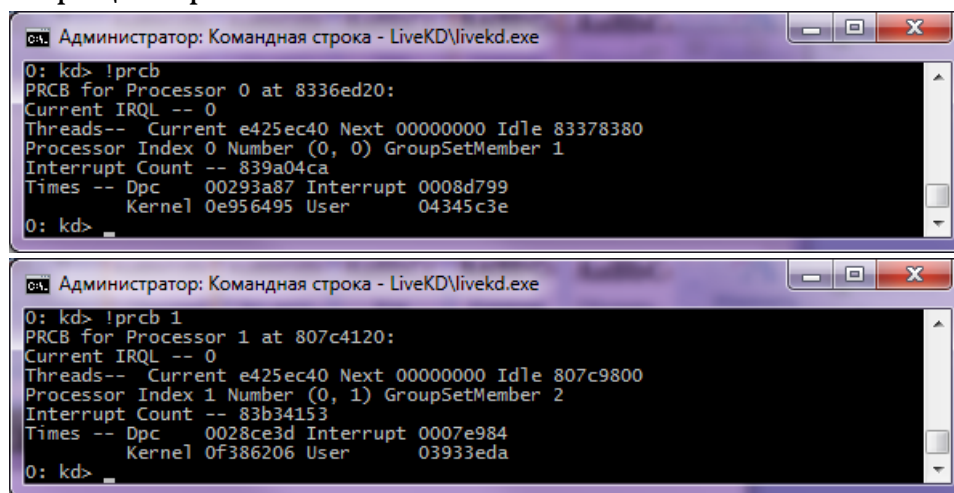
Получите список потоков выбранного процесса, выберите несколько потоков процесса и получите для них структуры ETHREAD. Запротоколируйте результаты в отчет с комментариями.

5. Подготовьте итоговый отчет с развернутыми выводами по заданию.

**Задание 5.2.** Исследовать регистр контроля процессора и очередь потоков готовых для выполнения.

**Указания к выполнению.**

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



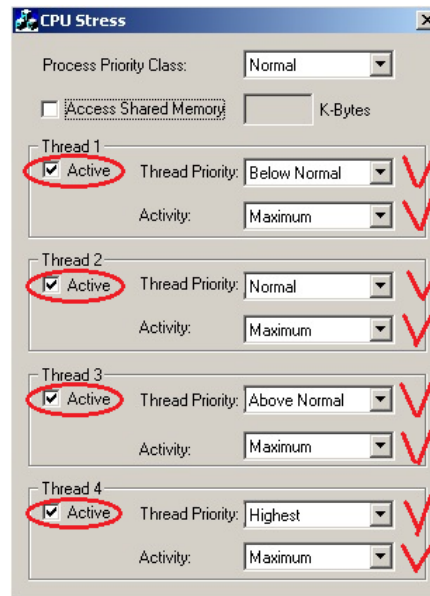
```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> !prcb
PRCB for Processor 0 at 8336ed20:
Current IRQ -- 0
Threads-- Current e425ec40 Next 00000000 Idle 83378380
Processor Index 0 Number (0, 0) GroupSetMember 1
Interrupt Count -- 839a04ca
Times -- Dpc 00293a87 Interrupt 0008d799
Kernel 0e956495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

Администратор: Командная строка - LiveKD\livekd.exe
0: kd> !prcb 1
PRCB for Processor 1 at 807c4120:
Current IRQ -- 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>
```

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. Важно обеспечить загрузку системы несколькими потоками, желательно с разными приоритетами.



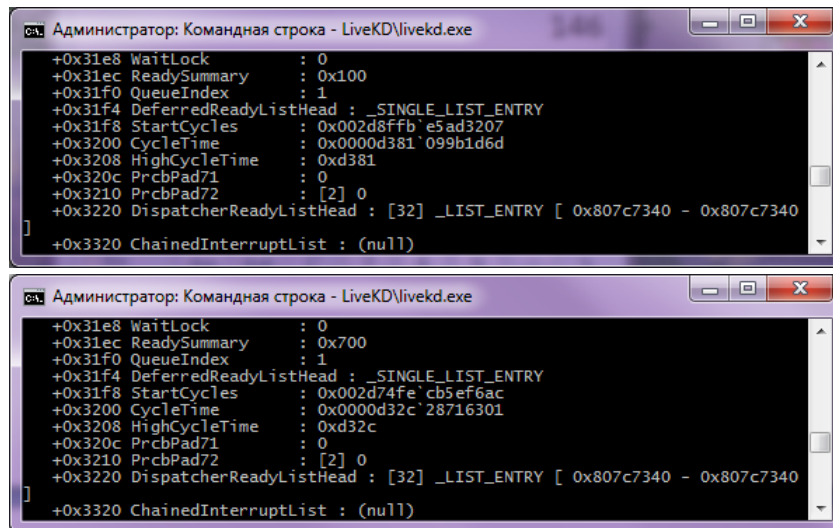
3. Для получения структуры KPRCB (Kernel Processor Register Control Block) для выбранного процессора вызовите команду `dt _kprcb <адрес PRCB>`. Например, для процессора «0» из примера предыдущего пункта `dt _kprcb 8336ed20`.

```

Администратор: Командная строка - LiveKD\livekd.exe
0: kd> dt nt!_kprcb 8336ed20
+0x000 MinorVersion : 1
+0x002 MajorVersion : 1
+0x004 CurrentThread : 0x83378380 _KTHREAD
+0x008 NextThread : (null)
+0x00c IdleThread : 0x83378380 _KTHREAD
+0x010 LegacyNumber : 0 ''
+0x011 NestingLevel : 0 ''
+0x012 BuildType : 0 ''
+0x014 CpuType : 6 ''
+0x015 CpuID : 1 ''
+0x016 CpuStep : 0xd70a ''
+0x016 CpuStepping : 0xa ''
+0x017 CpuModel : 0x17 ''
+0x018 ProcessorState : _KPROCESSOR_STATE
+0x338 KernelReserved : [16] 0
+0x378 HalReserved : [16] 0xe1d100
+0x3b8 CFlushSize : 0x40
+0x3bc CoresPerPhysicalProcessor : 0x2 ''
+0x3bd LogicalProcessorsPerCore : 0x1 ''
+0x3be PrcbPad0 : [2] ""
  
```

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.





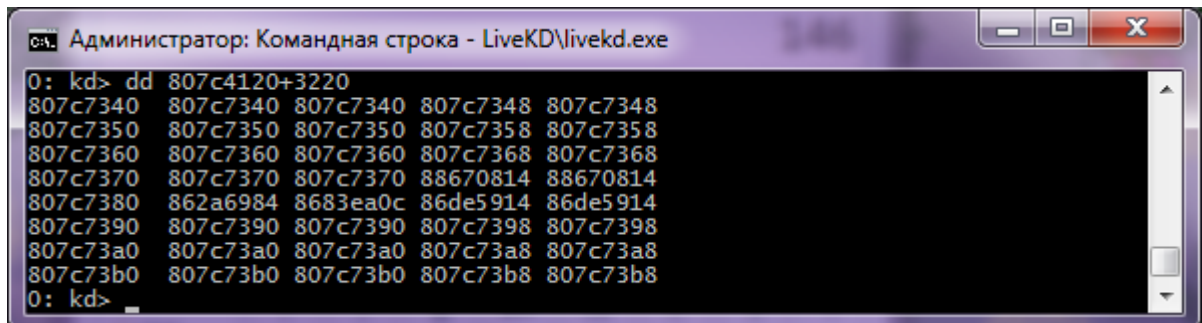
```
Администратор: Командная строка - LiveKD\livekd.exe
+0x31e8 WaitLock : 0
+0x31ec ReadySummary : 0x100
+0x31f0 QueueIndex : 1
+0x31f4 DeferredReadyListHead : _SINGLE_LIST_ENTRY
+0x31f8 StartCycles : 0x002d8ffb'e5ad3207
+0x3200 CycleTime : 0x0000d381'099b1d6d
+0x3208 HighCycleTime : 0xd381
+0x320c PrcbPad71 : 0
+0x3210 PrcbPad72 : [2] 0
+0x3220 DispatcherReadyListHead : [32] _LIST_ENTRY [ 0x807c7340 - 0x807c7340
]
+0x3320 ChainedInterruptList : (null)

Администратор: Командная строка - LiveKD\livekd.exe
+0x31e8 WaitLock : 0
+0x31ec ReadySummary : 0x700
+0x31f0 QueueIndex : 1
+0x31f4 DeferredReadyListHead : _SINGLE_LIST_ENTRY
+0x31f8 StartCycles : 0x002d74fe'cb5ef6ac
+0x3200 CycleTime : 0x0000d32c'28716301
+0x3208 HighCycleTime : 0xd32c
+0x320c PrcbPad71 : 0
+0x3210 PrcbPad72 : [2] 0
+0x3220 DispatcherReadyListHead : [32] _LIST_ENTRY [ 0x807c7340 - 0x807c7340
]
+0x3320 ChainedInterruptList : (null)
```

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.

4. Чтобы просмотреть содержимое очереди потоков процессора «1», введите в отладчике следующую команду: **dd 807c4120+3220**. Адрес получается путем прибавления смещения поля **DispatcherReadyListHead** (3220) к стартовому адресу структуры **KPRCB**.



```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> dd 807c4120+3220
807c7340 807c7340 807c7340 807c7348 807c7348
807c7350 807c7350 807c7350 807c7358 807c7358
807c7360 807c7360 807c7360 807c7368 807c7368
807c7370 807c7370 807c7370 88670814 88670814
807c7380 862a6984 8683ea0c 86de5914 86de5914
807c7390 807c7390 807c7390 807c7398 807c7398
807c73a0 807c73a0 807c73a0 807c73a8 807c73a8
807c73b0 807c73b0 807c73b0 807c73b8 807c73b8
0: kd>
```

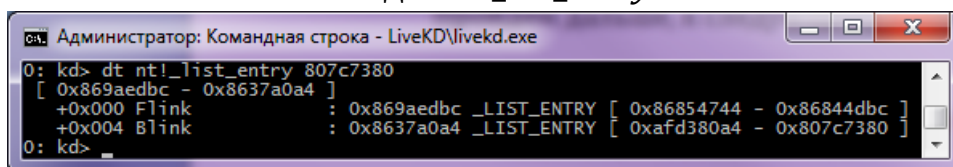
Тип **LIST\_ENTRY** описывает двунаправленный список и представляет собой структуру, состоящую из двух полей: **Flink** (Forward Link) – указатель на следующий элемент списка и **Blink** (Backward Link) – указатель на предыдущий элемент списка.

На рисунке показаны первые 16 элементов массива **DispatcherReadyListHead**, состоящие из двух адресов – **Flink** и **Blink**.

Большинство элементов массива описывают пустые списки – это ситуация, когда адреса в обоих полях структуры **LIST\_ENTRY** совпадают и

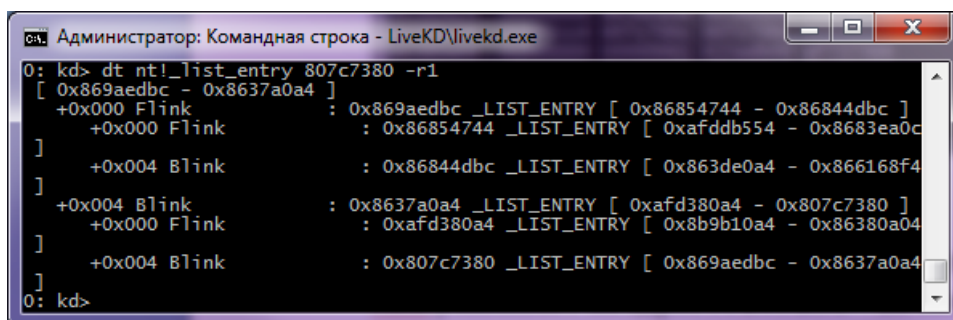
указывают на одно и то же поле – **Flink**. Например, на рисунке первый элемент массива представляет собой структуру `LIST_ENTRY`, располагающуюся по адресу `07c7340`, оба поля которой содержат тот же самый адрес.

Нас интересуют непустые списки – это восьмой, девятый и десятый элементы массива (см. единичные биты в поле **ReadySummary**). Рассмотрим восьмой элемент массива, описывающий очередь готовых потоков с максимальным в данный момент приоритетом. Чтобы определить следующий элемент очереди необходимо обратиться к ячейке, адрес которой соответствует **Flink** первого элемента, например, можно воспользоваться командой `dt _list_entry 807c7380`.



```
0: kd> dt nt!_list_entry 807c7380
[ 0x869aedbc - 0x8637a0a4 ]
+0x000 Flink      : 0x869aedbc _LIST_ENTRY [ 0x86854744 - 0x86844dbc ]
+0x004 Blink      : 0x8637a0a4 _LIST_ENTRY [ 0xafd380a4 - 0x807c7380 ]
0: kd>
```

Та же самая команда может работать в рекурсивном режиме, выводя n-количество вложенных записей.



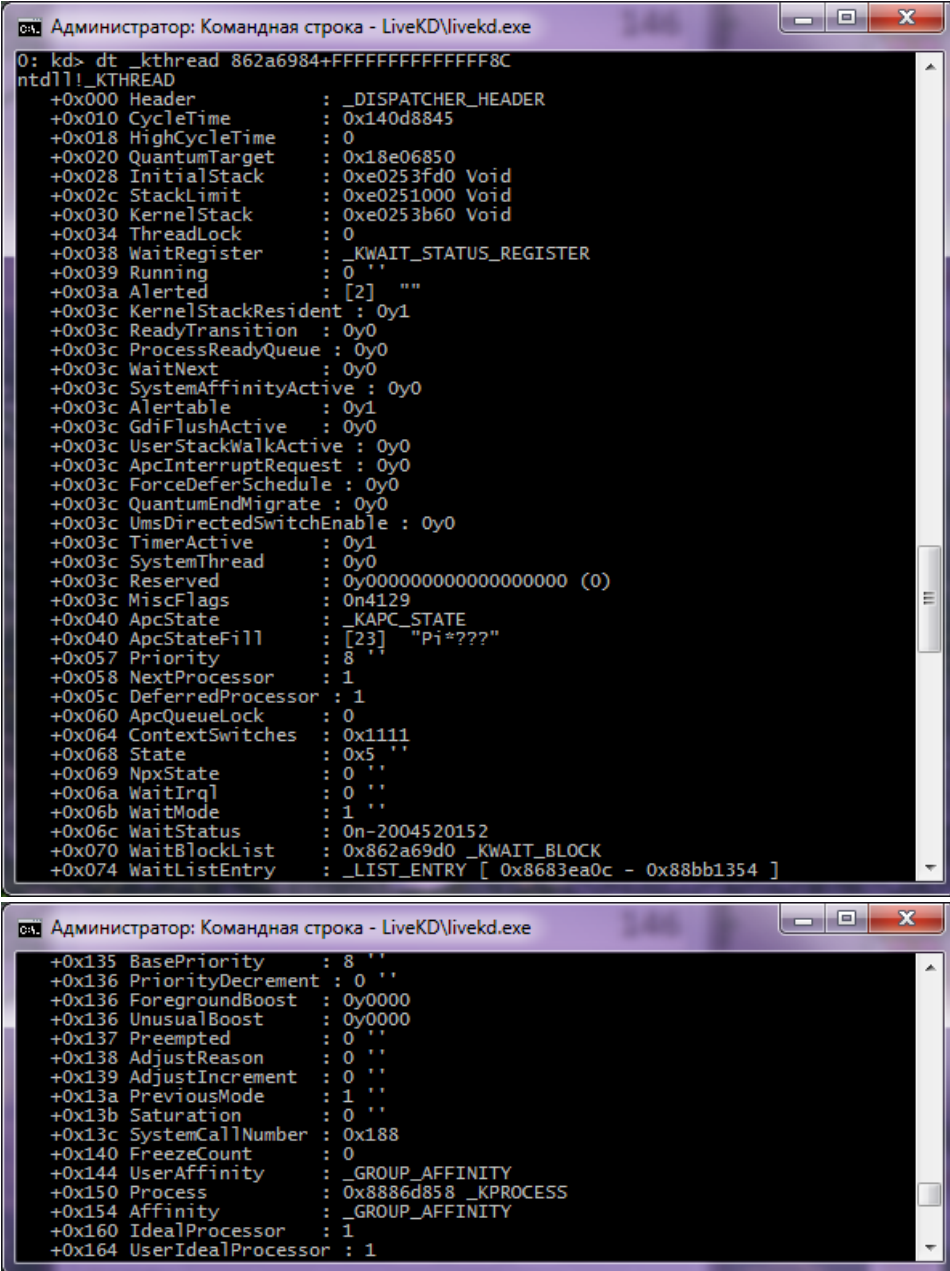
```
0: kd> dt nt!_list_entry 807c7380 -r1
[ 0x869aedbc - 0x8637a0a4 ]
+0x000 Flink      : 0x869aedbc _LIST_ENTRY [ 0x86854744 - 0x86844dbc ]
+0x000 Flink      : 0x86854744 _LIST_ENTRY [ 0xafddb554 - 0x8683ea0c ]
]
+0x004 Blink      : 0x86844dbc _LIST_ENTRY [ 0x863de0a4 - 0x866168f4 ]
]
+0x004 Blink      : 0x8637a0a4 _LIST_ENTRY [ 0xafd380a4 - 0x807c7380 ]
+0x000 Flink      : 0xafd380a4 _LIST_ENTRY [ 0x8b9b10a4 - 0x86380a04 ]
]
+0x004 Blink      : 0x807c7380 _LIST_ENTRY [ 0x869aedbc - 0x8637a0a4 ]
]
0: kd>
```

Определите очереди готовых потоков с максимальным в данный момент приоритетом для всех процессоров Вашей вычислительной системы. Запротоколируйте результаты в отчет.

5. По значению адреса структуры `LIST_ENTRY` можно определить адрес начала структуры `KTHREAD` соответствующего потока. Данное поле `LIST_ENTRY` располагается по смещению `0x074` (для MS Windows 7) относительно начала структуры `KTHREAD`, это можно узнать, введя команду: `dt _kthread`. Таким образом, чтобы узнать адрес начала структуры `KTHREAD` потока в очереди готовых потоков, нужно из адреса, по которому располагается элемент списка очереди, вычесть смещение `0x074` (или прибавить `0xFFFFFFFFFFFFFFF8C`). Выведем на экран структуру `KTHREAD` для первого потока в очереди потоков с приоритетом 8. Адрес соответствующего элемента списка равен



862a6984, поэтому используем команду: `dt _kthread 862a6984+0xFFFFFFFFFFFFFFF8C`.

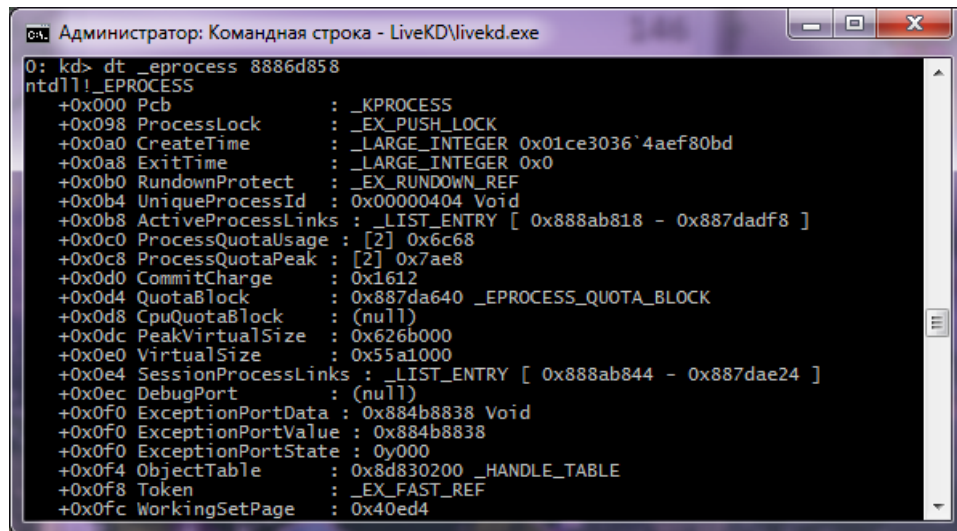


```

0: kd> dt _kthread 862a6984+0xFFFFFFFFFFFFFFF8C
ntdll!_KTHREAD
+0x000 Header : _DISPATCHER_HEADER
+0x010 CycleTime : 0x140d8845
+0x018 HighCycleTime : 0
+0x020 QuantumTarget : 0x18e06850
+0x028 InitialStack : 0xe0253fd0 Void
+0x02c StackLimit : 0xe0251000 Void
+0x030 KernelStack : 0xe0253b60 Void
+0x034 ThreadLock : 0
+0x038 WaitRegister : _KWAIT_STATUS_REGISTER
+0x039 Running : 0 ''
+0x03a Alerted : [2] ""
+0x03c KernelStackResident : 0y1
+0x03c ReadyTransition : 0y0
+0x03c ProcessReadyQueue : 0y0
+0x03c WaitNext : 0y0
+0x03c SystemAffinityActive : 0y0
+0x03c Alertable : 0y1
+0x03c GdiFlushActive : 0y0
+0x03c UserStackWalkActive : 0y0
+0x03c ApcInterruptRequest : 0y0
+0x03c ForceDeferSchedule : 0y0
+0x03c QuantumEndMigrate : 0y0
+0x03c UmsDirectedSwitchEnable : 0y0
+0x03c TimerActive : 0y1
+0x03c SystemThread : 0y0
+0x03c Reserved : 0y00000000000000000000 (0)
+0x03c MiscFlags : 0n4129
+0x040 ApcState : _KAPC_STATE
+0x040 ApcStateFill : [23] "Pi*???"
+0x057 Priority : 8 ''
+0x058 NextProcessor : 1
+0x05c DeferredProcessor : 1
+0x060 ApcQueueLock : 0
+0x064 ContextSwitches : 0x1111
+0x068 State : 0x5 ''
+0x069 NpxState : 0 ''
+0x06a WaitIrql : 0 ''
+0x06b WaitMode : 1 ''
+0x06c WaitStatus : 0n-2004520152
+0x070 WaitBlockList : 0x862a69d0 _KWAIT_BLOCK
+0x074 WaitListEntry : _LIST_ENTRY [ 0x8683ea0c - 0x88bb1354 ]

0: kd> dt _EPROCESS 862a6984+0xFFFFFFFFFFFFFFF8C
ntdll!_EPROCESS
+0x135 BasePriority : 8 ''
+0x136 PriorityDecrement : 0 ''
+0x136 ForegroundBoost : 0y0000
+0x136 UnusualBoost : 0y0000
+0x137 Preempted : 0 ''
+0x138 AdjustReason : 0 ''
+0x139 AdjustIncrement : 0 ''
+0x13a PreviousMode : 1 ''
+0x13b Saturation : 0 ''
+0x13c SystemCallNumber : 0x188
+0x140 FreezeCount : 0
+0x144 UserAffinity : _GROUP_AFFINITY
+0x150 Process : 0x8886d858 _KPROCESS
+0x154 Affinity : _GROUP_AFFINITY
+0x160 IdealProcessor : 1
+0x164 UserIdealProcessor : 1
  
```

Обратите внимание на поле **Priority** – там указан приоритет 8, который совпадает с приоритетом очереди потоков. Чтобы узнать процесс, к которому принадлежит данный поток, найдем поле **Process** структуры KTHREAD. Выведем на экран структуру EPROCESS по найденному адресу и узнаем имя исполняемого образа по полю **ImageFileName**.



```
Администратор: Командная строка - LiveKD\livekd.exe
0: kd> dt _eprocess 8886d858
ntdll!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x098 ProcessLock : _EX_PUSH_LOCK
+0x0a0 CreateTime : _LARGE_INTEGER 0x01ce3036'4aef80bd
+0x0a8 ExitTime : _LARGE_INTEGER 0x0
+0x0b0 RundownProtect : _EX_RUNDOWN_REF
+0x0b4 UniqueProcessId : 0x00000404 Void
+0x0b8 ActiveProcessLinks : _LIST_ENTRY [ 0x888ab818 - 0x887dadf8 ]
+0x0c0 ProcessQuotaUsage : [2] 0x6c68
+0x0c8 ProcessQuotaPeak : [2] 0x7ae8
+0x0d0 CommitCharge : 0x1612
+0x0d4 QuotaBlock : 0x887da640 _EPROCESS_QUOTA_BLOCK
+0x0d8 CpuQuotaBlock : (null)
+0x0dc PeakVirtualSize : 0x626b000
+0x0e0 VirtualSize : 0x55a1000
+0x0e4 SessionProcessLinks : _LIST_ENTRY [ 0x888ab844 - 0x887dae24 ]
+0x0ec DebugPort : (null)
+0x0f0 ExceptionPortData : 0x884b8838 Void
+0x0f0 ExceptionPortValue : 0x884b8838
+0x0f0 ExceptionPortState : 0y000
+0x0f4 ObjectTable : 0x8d830200 _HANDLE_TABLE
+0x0f8 Token : _EX_FAST_REF
+0x0fc WorkingSetPage : 0x40ed4
```

Как видно из рисунка, поток, находящийся первым (и единственным) в очереди готовых потоков с приоритетом 8, принадлежит процессу *svchost.exe*. Именно этот поток в данный момент будет выбран на выполнение.

Определите процессы-владельцев всех готовых потоков с максимальным в данный момент приоритетом 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.

**Указания к выполнению.**

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

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

где N=10000000.

- Используйте распределение итераций блоками (размер блока = 10 \* N<sub>студбилета</sub>) 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 передачи потокам параметров назначенных блоков итераций используйте механизм TLS (нечетные номера студенческого билета – статический TLS, четные номера студенческого 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.

**Указания к выполнению.**

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

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

где N=10000000.

- Распределите работу по потокам с помощью OpenMP-директивы **for**.
  - Используйте динамическое планирование блоками итераций (размер блока = 10 \* N<sub>студбилета</sub>).
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 результатами прошлой работы.
6. Подготовьте итоговый отчет с развернутыми выводами по заданию.