滴水逆向培训

基础教程

昆山滴水信息技术有限公司

（内部学习资料）

地址：昆山市巴城镇学院路 88 号
邮编：215311
TEL：0512-57882866
官网：www.dtdishui.com
论坛：www.dtdebug.com
EMAIL：kunshandishui@163.COM

目录

第一章 进制

引言：

进制跟我们生活是息息相关的，比如时钟，星期等，计算机也离不开进制，计算机是通过二进制数进行操作和运算的。

我们为什么要学习进制？

方便我们了解计算机，了解计算机的运行，为以后的学习打下基础。

什么才是正确的学习方法：

忘掉呆板的十进制！说到进制，其时大家都会，只是生活中的运用把其它的进制都丢弃了，只留下十进制，这一章主要是带我们了解各种进制，找回应有的记忆。

本章必须要掌握的知识点：

1．各种进制的书写方法。
2．进制间的运算。
3．计算机中负数的表示方法。
4．布尔代数。

本章常犯的错误：

1．总是以十进制为依托去考虑其他进制。
2．其他进制判断大小时先转换成十进制数。
3．负数与符号位。

1.1 数据进制

本节主要内容：

1．了解进制。
2．各种进制的书写方法。

老师语录：

现在请一个同学上来写出 10 进制的 0-100
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39

40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
100

大家看着上面的的数字说一句话：

小陶 say：有 101 个数。

小胡 say：十进制数从 0 到 100。

……

其实这就是小学入学考试，只要写出 0-100 就可以参加学习我们的课程。

下面我来给十进制下个定义：

十进制是由 0、1、2、3、4、5、6、7、8、9 十个符号组成，逢十进一。

你们给九进制下个定义：

九进制定义：九进制是由 0、1、2、3、4、5、6、7、8 九个符号组成，最小是 0，最大是 8，逢九进一。

练习：

用九进制写出十进制的 101 个元素：

0 1 2 3 4 5 6 7 8
10 11 12 13 14 15 16 17 18
20 21 22 23 24 25 26 27 28
30 31 32 33 34 35 36 37 38
40 41 42 43 44 45 46 47 48
50 51 52 53 54 55 56 57 58
60 61 62 63 64 65 66 67 68
70 71 72 73 74 75 76 77 78
80 81 82 83 84 85 86 87 88
100 101 102 103 104 105 106 107 108
110 111 112 113 114 115 116 117 118
120 121

现在给七进制下个定义：

七进制是由 0、1、2、3、4、5、6 七个符号组成，最小是 0，最大是 6，逢七进一。

练习：

用七进制写出十进制的 101 个元素：

0 1 2 3 4 5 6
10 11 12 13 14 15 16
20 21 22 23 24 25 26
30 31 32 33 34 35 36
40 41 42 43 44 45 46
50 51 52 53 54 55 56
60 61 62 63 64 65 66

100 101 102 103 104 105 106
110 111 112 113 114 115 116
120 121 122 123 124 125 126
130 131 132 133 134 135 136
140 141 142 143 144 145 146
150 151 152 153 154 155 156
160 161 162 163 164 165 166
200 201 202

现在给十一进制下个定义：

十一进制是由 0、1、2、3、4、5、6、7、8、9 还差一个符号，用 X 也行，用 A 也行，共十一个符号组成，最小是 0，最大是 X（或 A），逢十一进一。

练习：

用十一进制写出十进制的 101 个元素：

0 1 2 3 4 5 6 7 8 9 A
10 11 12 13 14 15 16 17 18 19 1A
20 21 22 23 24 25 26 27 28 29 2A
30 31 32 33 34 35 36 37 38 39 3A
40 41 42 43 44 45 46 47 48 49 4A
50 51 52 53 54 55 56 57 58 59 5A
60 61 62 63 64 65 66 67 68 69 6A
70 71 72 73 74 75 76 77 78 79 7A
80 81 82 83 84 85 86 87 88 89 8A
90 91

现在给三进制下个定义：三进制是由 0、1、2 共三个符号组成，最小是 0，最大是 2，逢三进一。

练习：

用三进制写出十进制的 101 个元素：

0 1 2
10 11 12
20 21 22
100 101 102
110 111 112
120 121 122
200 201 202
210 211 212
220 221 222
1000 1001 1002
1010 1011 1012
1020 1021 1022
1100 1101 1102
1110 1111 1112
1120 1121 1122

1200 1201 1202
1210 1211 1212
1220 1221 1222
2000 2001 2002
2010 2011 2012
2020 2021 2022
2100 2101 2102
2110 2111 2112
2120 2121 2122
2200 2201 2202
2210 2211 2212
2220 2221 2222
10000 10001 10002
10010 10011 10012
10020 10021 10022
10100 10101 10102
10110 10111 10112
10120 10121 10122
10200 10201

其实学计算机很简单，就是扳着手指头数数。如果不是数出来的，而是算出来的，那肯定是错误的想法。数的本质是数出来的。

比如：2+2 = 4 我们可以用手指头一个一个数出来。

课后理解： 进制其实是 N 种符号组成的。

各种进制的表示方法，如表 1-1 所示：

进制	实例
1	0，00，000，0000，00000，000000…
2	0，1，10，11，100，101，110，111，1000…
3	0，1，2，10，11，12，20，21，22，100…
4	0，1，2，3，10，11，12，13，20，21，22…
5	0，1，2，3，4，10，11，12，13，14，20…
6	0，1，2，3，4，5，10，11，12，13，14…
7	0，1，2，3，4，5，6，10，11，12，13，14…
8	0，1，2，3，4，5，6，7，10，11，12，13…
9	0，1，2，3，4，5，6，7，8，10，11，12…
10	0，1，2，3，4，5，6，7，8，9，10，11，12…
11	0，1，2，3，4，5，6，7，8，9，A，10，11…

表 1-1：1-11 进制的表示方法

课后疑问：

本节没有疑问。

课后总结：

进制是由元素组成的，N 进制就是有 N 个元素组成，逢 N 进一

课后练习：

1．在纸上用 1 到 16 进制分别描述 100 个数。
2．写一到十六进制，每进制 0-99。
3．0 到 16 进制，每个进制写 100 个数。

1.2 进制运算

本节主要内容：

1．二进制的好处。
2．进制间的运算。

老师语录：

十进制是大家小学时就会的：
0，1，2，3，4，5，6，7，8，9
那么九进制大家也应该都会：
0，1，2，3，4，5，6，7，8
十进制可以加减乘除，那么九进制照样可以加减乘除，直接算出结果，十一进制也可以。既然小于十进制可以加减乘除，大于十进制可以加减乘除，那就是说 N 进制都可以加减乘除。难道 N 进制都可以加减乘除，唯独二进制不可以加减乘除吗？

既然你们都会的话，书上为什么还要教二进制呢？还用进制间转换吗？你们可以看一下书上，有十进制转二进制，二进制转十进制，不是多此一举吗？三进制可以直接算出结果，九进制也可以直接算出结果，为什么二进制不可以呢？任何一种进制，它自身就是一个完美的体系结构，直接能加减乘除算术逻辑运算。

练习：

九进制加法：
7+8=16
九进制根本不需要去转换，可以直接算出结果，十进制也是，和其他进制没有关系，自成一个体系结构。任何进制都可以直接加减乘除。那么，同样的，三进制、二进制、八进制、十六进制都是一个完美的体系结构，都可以自己加减乘除，直接算出结果，如果谁去转换成二进制，那说明不懂。二进制可以直接加减乘除算结果，把结果拿来用就可以了。比如：你说你有“110”块钱不行吗？非要用十进制的吗？如果需要加减，可以直接加减，九进制可以直接加减，为什么二进制不能呢？既然加减乘除都可以，为什么还要去转换呢？现在懂了没有？大家理解所有进制了吗？
小桃 say：“懂了。”
小胡 say：“会了。”
二进制就是由 0 和 1 共两个符号组成，最小是 0，最大是 1，逢二进一，就学完了。
那现在我们给 N 进制下个定义：
N 进制就是由 0、1、2、3、4……N-1 共 N 个符号组成，逢 N 进一。
我们说过了，人类只会数数，不会加减，所以说我们用进制不能用的太大，因为十进制有十个符号，相互之间加减是我们能记住的。二进制就更简单，只有两个符号，两个之间相互加减，都是可以记住的，为什么计算机用二进制？因为二进制最简单，需要记住的东西最少，人类不会计算，当然计算机更不会计算。所以说他只能用最简单的记忆方法来算，因为十进制需要记住的符号太多了。

注：电子计算机只有高电平和低电平两种状态，所以只能使用二进制 0 和 1 来表示，并进制算术逻辑运算。

比如在十进制中，5+4=9，2+7=9，1+8=9，3+6=9，2+8=10，两两相加，符号太多了，很麻烦，相比较而言，二进制要简单的多。严格的说，他只有两个结果 0 和 1，所以非常简单。而其他进制太复杂，三进制，四进制，五进制，六进制都很复杂，所以说基本都不用。人类最常用的是十进制，因为人类有十个手指头，数起来很方便。最方便的数数工具，就是手指。人类天生就会使用十进制，当然也有民族使用五进制的，因为五进制数起来也很方便，一个手五个手指头，一个手专门去数。现在大家都懂了进制了吧？

练习：

用一百进制从 0 写到十进制的 101？现在大家自己在纸上写，或者谁上来写。

小刘 say：“后面咋整”？

老唐 say：“随便你，不是学过了吗？九进制会了，十进制会了，十一进制会了，也就是说大于十进制的都会。小于十进制的都会。难道就不会一百进制吗” ？

小李 say：“用什么来表示” ？

老唐 say：“随便”。

第八章 操作系统

引言：

操作系统是连接硬件与用户层软件的枢纽。

我们为什么要学习操作系统？

通过操作系统可以了解内核，从而熟悉计算机的架构。

本章必须要掌握的知识点：

1．进程与线程。
2．APC 机制。
3．事件等待。
4．异常。
5．内存管理。
6．句柄表。
7．软件调试。

8.1 进程结构体

本节主要内容：

1．eprocess 结构体成员介绍。
2．kprocess 结构体成员介绍。
3．进程链表。
4．遍历进程。

老师语录：

每一个进程都有一个 CR3，每个进程在 2G 以上都有一个结构体：EProcess。使用 WinDbg 查看如下（只显示部分内容）：

```
+0x000 Pcb             : _KPROCESS
+0x084 UniqueProcessId  : Ptr32 Void
+0x088 ActiveProcessLinks : _LIST_ENTRY
+0x0bc DebugPort        : Ptr32 Void
+0x11c VadRoot          : Ptr32 Void
+0x174 ImageFileName    : [16] UChar //16 个字节，进程名。
+0x190 ThreadListHead   : _LIST_ENTRY
+0x1b0 Peb             : Ptr32 _PEB
```

最前面有一个子结构体 KProcess，定义如下（只显示部分内容）：

```
+0x000 Header          : _DISPATCHER_HEADER
+0x018 DirectoryTableBase : [2] Uint4B// 8 个字节（32 位操作系统只使用了四个字节），存储 CR3 的值。
+0x020 LdtDescriptor    : _KGDTENTRY
+0x028 Int21Descriptor  : _KIDTENTRY
+0x030 IopmOffset       : Uint2B
+0x032 Iopl            : UChar
+0x034 ActiveProcessors : Uint4B
+0x040 ReadyListHead    : _LIST_ENTRY
+0x048 SwapListEntry    : _SINGLE_LIST_ENTRY
+0x050 ThreadListHead   : _LIST_ENTRY
+0x05c Affinity         : Uint4B
```

通过以下方式观察实例：

EProcess 数量：**!process 0 0** 指令显示所有进程结构体，显示的第一个数为结构体 EPROCESS 地址，然后使用命令 **dt EProcess 地址** 观察具体值。

例如：

```
kd> !process 0 0
PROCESS 81fb9830  SessionId: none  Cid: 0004    Peb: 00000000
Image: System
...
PROCESS 81c8f308  SessionId: 0  Cid: 078c    Peb: 7ffdd000
Image: notepad.exe
kd> dt _EPROCESS 81c8f308
+0x000 Pcb             : _KPROCESS
+0x084 UniqueProcessId  : 0x0000078c Void
+0x088 ActiveProcessLinks: _LIST_ENTRY[0x80562358 - 0x81b7f548]
+0x0bc DebugPort        : (null)
+0x11c VadRoot          : 0x81f37be8 Void
+0x174 ImageFileName    : [16]  "notepad.exe"
+0x190 ThreadListHead : _LIST_ENTRY [ 0x8194885c - 0x8194885c ]
+0x1b0 Peb             : 0x7ffdd000 _PEB
```

练习：

创建或打开多个应用程序，观察结构体变化。

_EPROCESS 0x88：ActiveProcessLinks : 8 个字节结构体，共两个成员：第一个成员指向前一个结构体，第二个成员指向后一个结构体，将所有进程围成一个圈（双向循环链表）。如图 8-1：

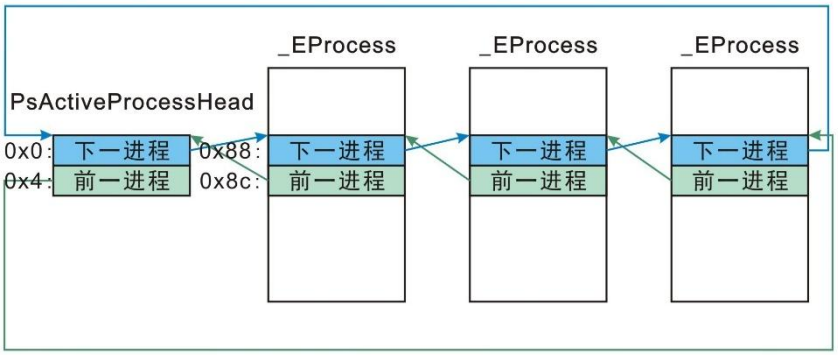

图 8-1：进程链表

注：指向的位置不是结构体头部，而是结构体中的成员 ActiveProcessLinks。

指向第一个进程 ActiveProcessLinks 结构体的是一个全局变量：**PsActiveProcessHead**，可以使用 WinDbg 观察。

如果将第一个进程结构体的链表中下一进程指向第三个进程，第三个进程结构体的链表中前一进程指向第一个进程，则进程 2 被隐藏（任务管理器中无法显示）。

```
kd> dd PsActiveProcessHead
8055b358  8a52c8b8 8a2878b0 00000001 b0fdca6c
8055b368  00000000 00040001 00000000 8055b374
8055b378  8055b374 00000000 7c920000 00000000
8055b388  00000000 00000000 00000000 00000000
8055b398  80528816 00000000 00000000 00000000
8055b3a8  8a0e55f8 89bff040 00000000 00000000
8055b3b8  00000000 00000000 00000001 ba4fbd50
8055b3c8  00000000 00040001 00000000 8055b3d4

kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 8a52c830  SessionId: none  Cid: 0004    Peb: 00000000  P
arentCid: 0000
    DirBase: 0035a000  ObjectTable: e1000c98  HandleCount: 328.
    Image: System  //第一个进程

8a52c8b8 - 88 =  8a52c830
```

练习：

打开计算器，并在任务管理器中隐藏该进程。

前一个：0x89671890 - 0x8958a9f0

计算器：0x8966dc78 - 0x8953a738

后一个：0x805638b8 - 0x89671890

修改为：

前一个：0x8966dc78 - 0x8958a9f0

后一个：0x805638b8 - 0x8953a738

如果我们不知道 PsActiveProcessHead，如何遍历，如何得到首进程？

说明：可以随机遍历，得到进程结构体后，减去 0x88，校验 DirectoryTableBase 是否有效。如果无效则是假进程（也就是 PsActiveProcessHead）。或者使用 ImageFileName 校验，具体如下：

```
kd> !list 0x81f0be28
Image: alg.exe
81f0be28  81c73e28 8200e2e8 000016b0 0001179c
wmiprvse.exe
81c73e28  8055b158 81f0be28 00001540 00014ab4
PsActiveProcessHead//可以知道其 CR3 无效
8055b158  821b98b8 81c73e28 00000001 b25c5a6c
System//使用!process 0 0 查看的第一个进程
821b98b8  81d51770 8055b158 00000000 00000000
smss.exe
81d51770  81da67a8 821b98b8 00000280 0000179c
csrss.exe
81da67a8  81fb4678 81d51770 00001980 000213c4
winlogon.exe
81fb4678  820fd5c8 81da67a8 00009c30 000193ac
services.exe
820fd5c8  81d54d30 81fb4678 00001720 0000887c
lsass.exe
81d54d30  81c94e28 820fd5c8 00002678 00012724
vmacthlp.exe
81c94e28  82078620 81d54d30 000007a8 00008444
svchost.exe
82078620  81cba7c8 81c94e28 00001a30 0001394c
svchost.exe
81cba7c8  81c4fe28 82078620 00003528 000122ac
svchost.exe
81c4fe28  81c45350 81cba7c8 00008a88 0002a8a4
svchost.exe
81c45350  81b15318 81c4fe28 00000fe0 0000df64
svchost.exe
81b15318  821373e8 81c45350 00001900 00012a1c
explorer.exe
821373e8  81dc2e28 81b15318 00002df0 000211b4
rundll32.exe
81dc2e28  81ce9240 821373e8 00001990 000145e4
spoolsv.exe
81ce9240  81cac810 81dc2e28 00001830 000144a4
VMwareTray.exe
81cac810  81f030a8 81ce9240 000009d8 0000ebfc
VMwareUser.exe
```

```
81f030a8  81cdbe28 81cac810 00000e10 00010894
ctfmon.exe
81cdbe28  81efce28 81f030a8 00000e60 0000f394
VMwareService.exe
81efce28  81ce4730 81cdbe28 00000f78 0000f83c
svchost.exe
81ce4730  8200e2e8 81efce28 00001068 0000f864
wuauclt.exe
8200e2e8  81f0be28 81ce4730 00001f68 0001d054
```

_EPROCESS 0xbc: DebugPort。和调试相关，没有被 attach（附加）的进程值为 0，反之不为 0。

练习：

使用 OD 附加（attach）一个进程，再将其 DebugPort 值清零，观察调试（F7）效果。

课后理解：

KPROCESS 和 EPROCESS 两个结构体成员的偏移是连续的，这两个结构体是继承关系。

课后疑问：

以删除进程中链表的方式隐藏进程，被隐藏的进程会正常运行吗？
不受影响，会正常运行。

课后总结：

系统中的所有进程都被链表链接起来。

课后练习

打开记事本，并在任务管理器中隐藏该进程。

8.2 线程结构体

本节主要内容：

1. ETHREAD 结构体。
2. KTHREAD 结构体。
3. 线程链表。

老师语录：

每个进程结构体中都包含线程信息：_EPROCESS 0x50：**ThreadListHead**。

每个线程对应一个结构体 ETHREAD，ETHREAD 头部是 KTHREAD 结构体。

ETHREAD 结构体可以使用 windbg 查看如下（只显示部分内容）：

```
+0x000 Tcb              : _KTHREAD
+0x1ec Cid              : _CLIENT_ID//8 个字节，两个成员，第一个成员是进程（管理该线程的进程）ID 号，第二个成员是线程 ID 号。
+0x220 ThreadsProcess   : Ptr32 _EPROCESS//指向当前进程
+0x224 StartAddress     : Ptr32 Void
+0x228 Win32StartAddress : Ptr32 Void
+0x22c ThreadListEntry  : _LIST_ENTRY
```

ETHREAD 第一个成员是 KTHREAD，结构如下（只显示部分）：

```
+0x000 Header           : _DISPATCHER_HEADER
+0x018 InitialStack     : Ptr32 Void
+0x01c StackLimit       : Ptr32 Void
+0x020 Teb              : Ptr32 Void
+0x028 KernelStack      : Ptr32 Void
+0x02c DebugActive      : UChar //一个字节，和调试相关（硬件断点）。
+0x034 ApcState         : _KAPC_STATE
+0x05c WaitBlockList    : Ptr32 _KWAIT_BLOCK
+0x060 WaitListEntry    : _LIST_ENTRY
+0x070 WaitBlock        : [4] _KWAIT_BLOCK
+0x0e0 ServiceTable     : Ptr32 Void
+0x118 QueueListEntry   : _LIST_ENTRY
+0x134 TrapFrame        : Ptr32 _KTRAP_FRAME
+0x138 ApcStatePointer  : [2] Ptr32 _KAPC_STATE
+0x140 PreviousMode     : Char
+0x14c SavedApcState    : _KAPC_STATE
+0x164 Alertable        : UChar
+0x165 ApcStateIndex    : UChar
+0x168 StackBase        : Ptr32 Void
+0x16c SuspendApc       : _KAPC
+0x1b0 ThreadListEntry  : _LIST_ENTRY
```

偏移 0x34 处为 ApcState 结构体：

```
struct _KAPC_STATE
{
  0x34:_LIST_ENTRY ApcListHead[2];
  0x44:_KPROCESS* Process;//指向进程
  ...
}
```

ThreadListHead 并非指向 ETHREAD 结构体首地址，而是偏移结构体首地址的 0x1b0（在 KTHREAD 结构体中）位置，并围成一个圈（双向循环链表），如图 8-2：

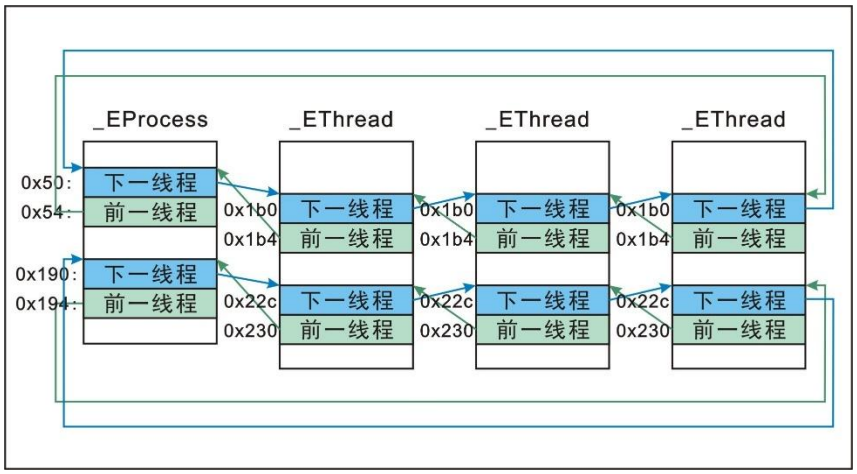

图 8-2：线程链表

练习：

写一个程序，在程序中创建多个线程，观察其链表圈。

cpu 执行多任务的关键是时钟，分配给每个线程或任务一定时间，时间结束，自动执行一个 int 指令保存栈并切换到其他任务（或线程）。

kthread 中最重要的成员是栈：

0x18：InitialStack

0x1c：StackLimit

0x28：KernelStack

程序切换时只有栈切换，没有寄存器，为什么？（参照老唐模拟线程切换代码，此代码在后续章节会介绍，如需要可在滴水论坛下载）

线程间切换使用 INT 指令实现，当然是切换 esp（堆栈），对于一个 cpu，只有一个线程 ID，只有一个上下文（CONTEXT），只能跑一段代码，为什么很多程序可以同时运行？

当执行线程的时候，我们可能会调 API，比如：GetMessage。如果一个线程不调用 API 且不退出，观察他是否会被切换（cpu 会占用 100%）。

当调用 API 时，操作系统执行 int 0x2E 进入 0 环，最终一定会调用一个检测是否切换线程的子函数，如果需要切换，会调用切换函数：至少需要三个成员：栈顶，栈底，栈当前位置。KTHREAD 中含有这三个成员：InitialStack、StackLimit、KernelStack。

线程切换时寄存器值不在结构体里面，而是保存在堆栈里面（包括 EIP）。

练习：

查看 Kthread 实例，在进程中创建（createThread）十个线程（功能可以部分相同），观察功能相同线程之间和功能不同线程之间的栈顶，栈底，栈当前位置、线程 ID、进程 ID、开始地址的相同和不同点并记录下来。

注：Ethread.**StartAddress** 0x224

Ethread. **Win32StartAddress** 0x228

练习：

看模拟线程切换代码，观察切换到其他线程后运行的第一行代码在哪里。

在查看线程结构体信息时，不能依赖于线程的起始地址（会被恶意修改）。

课后理解：

一个进程可以对应多个线程，一个线程只对应一个进程。

课后疑问：

EThread 结构体中 0x1b0 和 0x22c 的区别是什么？
没有区别。

课后总结：

线程切换时寄存器值不在结构体里面，而是保存在堆栈里面（包括 EIP）。

课后练习

一个线程只能做一件事，如何实现一个线程可以同时做多件事？
自己实现一个线程模拟多线程同时完成多项任务（参照模拟线程切换代码）。