Smalltalk/V – Win32 API call primitives

The api call is being done via primitive 94. The first literal in the compiled method must be the name of the exported API function. The second literal must be a byte array in a certain format. The first 4 bytes are being used to cache the address of the API function. The next byte is the number of arguments of the api function plus one (the return type). Accordingly the next bytes define the argument types (int, short, struct a.s.o).

This is the stack snapshot from beginning of the primitive to the actual api call.

instruction	what does it mean	comment
		higher addresses
		<-exit stack ptr points here
	(stack frame for api call method)	
call prim 94	return address – caller of primitive 94 (codeCache)	
push eax	ptr expected by GC	
push eax	ptr expected by GC	
push eax	receiver (the DLL)	
push edi	method context	
push ecx	api call method	
push edx	api frame link	
push ebp	old frame ptr	
push ebx	peek count	<- EBP points here
push eax	api proc address – used by "call [ebp-4]"	
push ebx	structPktChainHead	points to first structPkt ->
push ebx	return type	
push esi	exit stack ptr	
call pushArgs	return address – caller of pushArgs	
	stack buffer start (receives ESP-8)	
	stack buffer end (receives ESP-8 - stackbuffer 256/4352 bytes)	
	byte copy of smalltalk arg #1 (if fits on stack)	
	1st struct pkt (12 byte structure)	<-structPktChainHead/start
	byte copy of smalltalk arg #2 (if fits on stack)	
	2nd strct pkt	
	....	<- stack buffer end points here
	address of object byte copy #1	
	address of object byte copy #2	
	....	
call [ebp-4]	return address – caller is prim 94	
		lower addresses

The function saves various status information on the stack (receiver, current method ...). Then the conversion of the arguments happens. The byte array in the literal frame tells, how to convert arguments. The #struct argument types will get a special treatment. A block of memory is reserved on the stack, that holds a linked list of small structs called "struct packet". The structure contains these fields:

pktLink	link to next struct packet
pktArgSize	number of bytes for this argument (incl. the strctPkt size) & flags (structOut, memory on heap)
pktArgPtrPtr	pointer to pointer of original smalltalk byte object

The reserved stack block will be used to hold the struct packets and copies of the Smalltalk objects, if they fit into the stack block. If they don't fit into the stack block, memory from the global heap will be allocated. The pktArgSize will contain a special flag in this case.

```
Example:

| systemTime |
systemTime := WinSystemTime new.
KernelLibrary getSystemTime: systemTime asParameter.

Setting a breakpoint at GetSystemTime and dumping the stack after
the breakpoint is being hit.

kernel32!GetSystemTime:
7c80176f 8bff            mov     edi,edi

>dd esp
0013e764  02026ca6 0013f85c 00200001 00000000

The return address into the VM that did the call.
>u 02026ca6-3
02026ca3 ff55fc           call    dword ptr [ebp-0x4]

The pointer to the argument buffer.
>dd 0013f85c
0013f85c  00000000 00000000 00000000 00000000

As you can see, the address of the buffer is in the same address
range as the current ESP.
```

```
>r ebp
ebp=0013f888

>dd ebp-40
0013f848  02012bb8 02053da0 00000000 8000001c
0013f858  0013f8b0 00000000 00000000 00000000
0013f868  00000000 0013e76c 0013f850 02026ca3
0013f878  0013f8b4 00000009 0013f850 7c80176f
0013f888  ffffedcf 0013f8c4 00000000 2228ae6c
0013f898  22049a78 2458f454 2458f454 2458f454
0013f8a8  00ac2f77 00ac2e0b 22049b0c 2458f454
0013f8b8  22049b00 22049810 02037060 0013f8dc
```

```
0013f888 Peek count
7c80176f address of api call
0013f850 structPkt chain head
00000009 api return type
0013f8b4 api method exit stack ptr
02026ca3 return address of function "pushArgs"
0013f850 stack buffer start
0013e76c stack buffer end
4x DWORD buffer for api call at address 13f85C, which is the
         address being passed to the API
         (copy of smalltalk object); see above
3x DWORD 1st struct pkt
          – 00000000 marks end of linked list
          – 8000001C 1C = sizeOf(struct pkt)+ sizeOf(buffer)
            MSB = 1 struct must be copied back
          – 0013f8b0 ptr to ptr of smalltalk object (22049b0c)
```

After returning from the api call the result is being converted according to the return type. If the struct packet chain head is not null, the list is walked and according to the flags of each structure the bytes are being copied back to the Smalltalk objects. If memory was allocated from the global heap it is being deallocated. At last the ESP is being set to api method exit stack ptr (EBP-16).

The DynamicLinkLibrary class method: #primitiveCall:proc:arguments:types:returns: uses primitive 95, which works a little different. It expects all information about the API call as arguments. The main difference is, that struct arguments are NOT copied like primitive 94 does.

Same breakpoint as above:

```
kernel32!GetSystemTime:
7c80176f 8bff            mov     edi,edi
> dd esp
0013f894  0202eccf 2201ca9c
> u 0202eccf - 3
0202eccc ff550c          call    dword ptr [ebp+0xc]
> dd 2201ca9c
2201ca9c  00000000 00000000 00000000 00000000
Pointer of argument is not in the address range of ESP.
```