What does the CreateProcess function do if there is no space between the program name and the arguments?

devblogs.microsoft.com/oldnewthing/20110808-00

August 8, 2011

Raymond Chen

In an old discussion of why the `CreateProcess` function modifies its command line, commenter Random832 asked, "What if there is no space between the program name and arguments – like "cmd/?" – where does it put the null then?"

The `CreateProcess` function requires a space between the program name and arguments. If you leave out the space, then the arguments are considered as part of the program name (and you'll almost certainly get `ERROR_FILE_NOT_FOUND` back).

It sounds like Random832 has confused `CreateProcess` command line parsing with `cmd.exe` command line parsing. Clearly the two parsers are different; you can see this even without playing with spaces between the program name and the arguments:

```
C:\>C:\Program Files\Windows NT\Accessories\wordpad.exe
'C:\Program' is not recognized as an internal or external command,
operable program or batch file.
```

If the command line had been parsed by `CreateProcess`, this would have succeeded in running the Wordpad program, because, as I noted in the original article, the `Create-Process` function modifies its command line in order to find where the program name ends and the command lines begin, an example of which can be found in the `CreateProcess` documentation. In this case, it would have plunked the null character into each of the spaces in the command line, finding that each one failed, until it finally tried treating the entire string as the program name, at which point it would have succeeded. The fact that it failed demonstrates that `CreateProcess` didn't do the parsing.

The `cmd.exe` program permits the space between a program name and its arguments to be elided if the arguments begin with a character not permitted in file names. Once it figures out what you're running, and it determines that what you're running is a program, it call the `CreateProcess` function with an explicit application and command line.

But you don't have to take my word for it. You can just see for yourself. (In fact, this is exactly what I did to investigate the issue in the first place.)

```
C:>ntsd -2 cmd.exe
```

Two windows will open, one for your debugger and one for `cmd.exe`. (You are welcome to replace `ntsd` with your favorite debugger. I chose `ntsd` because—at least until Windows XP—it came preinstalled, thereby avoiding multiplying the problem from one to two.)

In the debugger, set a breakpoint on `kernel32!CreateProcessW`, then resume execution. In the `cmd.exe` window, type `cmd/?`. The breakpoint will fire, and you can look at the parameters:

```
Breakpoint 0 hit
eax=0046f600 ebx=00000000 ecx=004f8de0 edx=00000000 esi=00000000 edi=00000001
eip=757820ba esp=0046f544 ebp=0046f704 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
kernel32!CreateProcessW:
757820ba 8bff            mov     edi,edi
0:000> dd esp l4
0046f544  4a5e3dd7 004f5420 004f8db0 00000000
0:000> du 004f5420
004f5420  "C:\Windows\system32\cmd.exe"
0:000> du 004f8db0
004f8db0  "cmd /?"
```

Observe that `cmd.exe` did its own manual path search to arrive at an executable of `C:\Windows\system32\cmd.exe`, and also that it secretly inserted a space between the `cmd` and the slash.

Raymond Chen

Follow