

# Requiem for the VFP ?

## I don't think so.



13 lat temu w 2012 roku oprogramowanie mojego autorstwa pracowało w dużych przedsiębiorstwach segmentu piekarskiego. Jest to oprogramowanie którego geneza sięga roku 1998r. Oprogramowanie to było napisane w języku c++ oraz Clipper 5.2E.

W tamtych czasach jedyną rozsądną dla mnie ( zachowałem bardzo dużo kodu źródłowego z Clipper ) alternatywą dla przenosin do świata 32bit był oczywiście Visual FoxPro firmy Microsoft. Język w zasadzie taki sam jak Clipper z bardzo niewielkimi zmianami. Bardzo szybko przekonałem się co to narzędzie potrafi jeśli chodzi o wydajność. Zarazem pojąłem dlaczego firma M\$ kupiła od Ashton-Tate FoxPro w 1992. Kupiła zaawansowaną matematykę tkwiącą w jego motorze bazodanowym, a którą teraz możemy podziwiać w kolejnych odsłonach MSSQL.



Jako że w 1998r nie było jeszcze MSQL natomiast SQL 7 firmy Microsoft ( pamięćcie to jeszcze ? ) wzbudzał we mnie tylko politowanie gdy testowałem jego wydajność, zdecydowałem się na VFP oraz pliki DBF/CDX. Już wtedy bariera 2GB na rozmiar pliku była dla mnie sporym problemem jednak miałem nadzieję, że w niedługim czasie firma M\$ złamie tę i inne bariery mocno doskwierające już wtedy VFP (brak możliwości budowania natywnych usług, brak wbudowanej wielowątkowości itd.). Miałem nadzieję, że w niedługim czasie firma VFP usunie te bariery. Jak wiemy nie usunęła a produkt ubiła w 2007 roku, tak jak wiele innych które kupiła po to by przejąć technologię a sam produkt zakończyć by udawać, że to ich osiągnięcie inżynierskie. Powyższe niedogodności zostały oczywiście rozwiązane przy użyciu bibliotek napisanych w c++ czyli vfp2c32. źródło: [Christian Ehlscheid https://github.com/ChristianEhlscheid/vfp2c32](https://github.com/ChristianEhlscheid/vfp2c32)

W roku 2012 doszedłem do ściany moich możliwości technologicznych jeśli chodzi o pliki DBF/CDX. Mój największy klient miał w tym okresie około 70 użytkowników pracujących w centrali na aplikacjach typu desktop. Oraz ponad 300 różnych stanowisk w sieci WAN, głównie POS, które również kilkadziesiąt razy na godzinę komunikowało się z centralą. Przeglądając obecne wtedy na rynku języki oprogramowania natknąłem się na kompilator niemieckiej firmy Alaska Software | Xbase++.

<https://www.alaska-software.com/>.



**Język ten bardzo podobny ( ta sama rodzina języków XBASE ) do VFP ma wiele zalet, że wymienię tylko niektóre z nich.**

- ✓ jest bardzo szybki i stabilny ( Alaska Software to bezpośrednia nakładka na c++ ).
- ✓ rewelacyjna wielowątkowość.
- ✓ bardzo mocne natywne wsparcie dla TCP/IP zestaw funkcji umożliwiających budowanie dowolnych usług TCP/IP oraz UDP.
- ✓ rewelacyjna współpraca z serwerami COM/DCOM VFP.
- ✓ bardzo dobre wsparcie dla Postgresql.

**Za pomocą tego narzędzia zbudowałem usługę TCP/IP która wykonuje każde polecenie VFP po stronie serwera. Czyli klient-serwer dla plików DBF/CDX ale nie tylko. Gdyż w chwili obecnej możliwa jest obsługa następujących motorów bazodanowych:**

✓ **DBF/CDX** ✓ MSSQL ✓ Postgresql ✓ Oracle ✓ MySql ✓ SQLite

Zapytania piszemy w SQL który znamy z VFP. W przypadku gdy w danym silniku bazodanym występują różnice w składni języka SQL, Foxx-i Server sam dokonuje translacji na daną odmianę języka SQL. Warunkiem jest jednakże 100% zgodność składni zapytań z VFP.

Dzięki temu zbudowałem aplikację która bez zmiany ani jednej linijki kodu może pracować na wszystkich wymienionych wyżej silnikach bazodanowych.

Usługa napisana w Alaska Software ( daemon TCP/IP ) ma za zadanie odbierać przychodzące połączenia z sieci lokalnej oraz WAN. Wątkami roboczymi są serwery COM Visual FoxPro i może ich być dowolna ilość.

Foxx-i Server sam sobie dobiera optymalną ilość wątków roboczych w zależności od obciążenia.

Po stronie serwera można wykonać dowolną instrukcję SELECT, UPDATE, DELETE ect. wywołać dowolną procedurę składowaną, bądź za pomocą natywnej funkcji VFP EXECSCRIPT() wykonać dowolny kod VFP z ekstremalną szybkością znaną z VFP po stronie serwera.

W 2019 roku w oparciu o Foxx-i Server zbudowałem swój własny model AI. Ada++.

Jako podstawową bazę danych dla moich sieci neuronowych wybrałem VFP z tego względu, że ani MSSQL ani Postgresql nie zapewniały mi odpowiedniej wydajności w zapytaniach które stosuję, a VFP tak. Dzięki temu jestem w stanie błyskawicznie generować prognozy sprzedaży na 7 dni do przodu uwzględniając 23 różnych kryteriów. U największego mojego klienta który ma kilkadziesiąt sklepów własnych estymacja która zawiera prognozę zamówień na 7 dni do przodu wykonuje się w niecałe 30 min i nie potrzeba do tego koparek NVIDIA. Daje mi to niesamowitą elastyczność np. w przypadku dynamicznie zmieniającej się prognozy pogody, świąt itd.

Algorytm ten w chwili obecnej samodzielnie zamawia towar dla sieci sklepów własnych o wartości **1 mln EUR dziennie**.

<https://infopiek.pl/en/offer/ada/>

oraz Ada++.AntiFraud. Algorytm który pilnuje by rozliczenia finansowe w sklepie zgadzały się.

<https://infopiek.pl/en/offer/ada-antifraud/>

Foxx-i Server w chwili obecnej zwraca dane z zapytań w następujących formatach:

✓ Foxx-i Server format

- ✓ JSON
- ✓ XML
- ✓ CSV
- ✓ DBF
- ✓ TXT
- ✓ XLS
- ✓ HTML



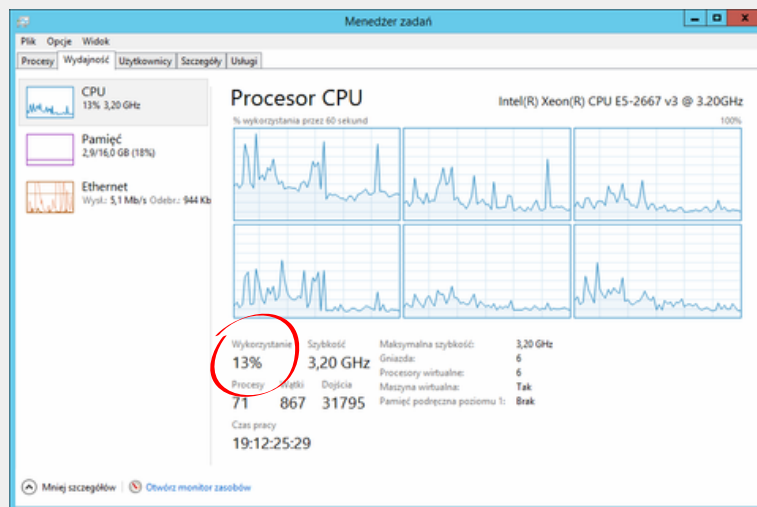
Poniżej zrzuty ekranu z produkcyjnej instalacji Foxx-i Server jednego z naszych klientów..

Srednia wydajność tego serwera wynosi 290 operacji zapisu/odczytu na sekundę.

Procesor Xeon(R) CPU E5-2667 v3 3.2 GHz, w chwili gdy piszę te słowa jest wart 10 USD a obciążenie procesora bardzo rzadko przekracza 23%.

```

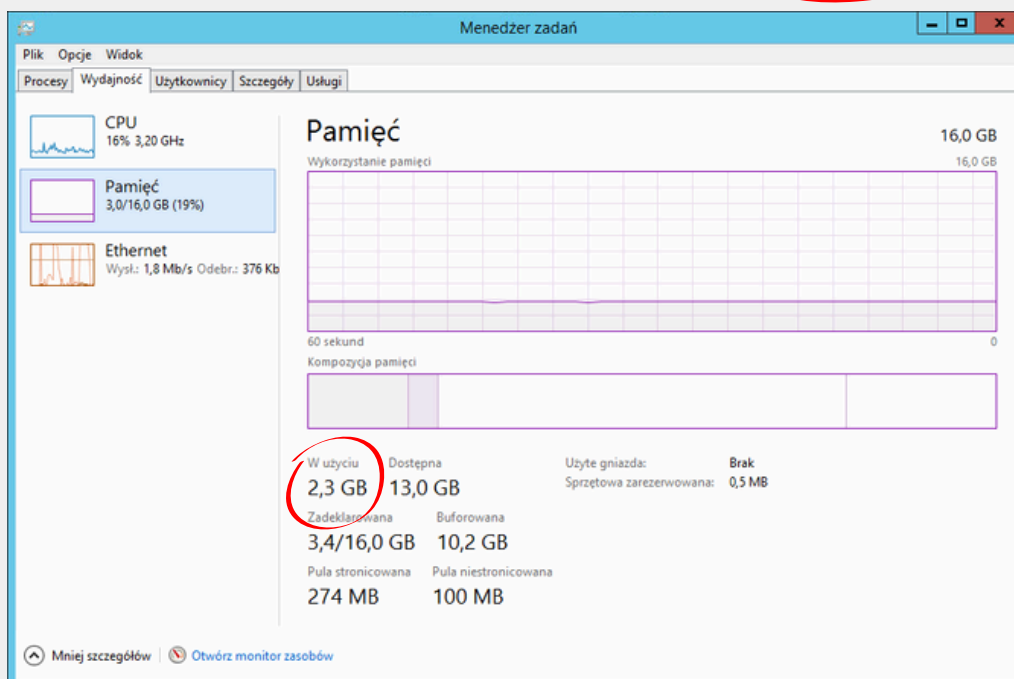
Process ID: 3236
Process Name: VikingSrvTcp.exe
WorkingSetSize: 32 Max: 46 69.18 %
PageFileUsage: 10 Max: 23 43.48 %
Virtual size ngb: 1.418 Max: 1.425 69.54 %
PageFaults: 760.174 Up to: 77
ThreadCount: 18 Max: 29 Min: 18
Processor time: 122.562500000000 sec 2.042708333333 min
Running from: 0 dni 02:47:41 SQL: 87 292 /RELIST: 0
DISKMEM SEND: 37 976 461 36.22 Mgb
DISKMEM COUNT: 6
SHAREDMEM_ERROR: 0
TOTAL = 1 176 461 536 1 121.96 Mgb 13 477 b/SQL
MAX STRING RCV: 1 338 418 1 307 kb
MAX STRING SEND: 1 527 550 1 497 kb
OPER PER SEC: ( 29 * 10 ) = 290 ops
INTERN SPEED: ( 29 ops
01 00712 00189 SERVICEAPP: START
03 01268 00373 SERVICEAPPLICATION: MAIN
04 01356 01198 CHECKCHILD5
05 01388 01198 CHECKCHILD5
06 01620 02395 VIKTHREAD: EXECUTE
07 01660 02395 VIKTHREAD: EXECUTE
08 01700 02395 VIKTHREAD: EXECUTE
09 01744 02395 VIKTHREAD: EXECUTE
10 01788 02395 VIKTHREAD: EXECUTE
11 01832 02395 VIKTHREAD: EXECUTE
12 01876 02395 VIKTHREAD: EXECUTE
13 01920 02395 VIKTHREAD: EXECUTE
14 01964 02395 VIKTHREAD: EXECUTE
  
```



Sama usługa Foxx-i Server zajmuje w pamięci serwera zaledwie 10 mgb RAM.

Nazwa	Identyfikator...	Stan	Nazwa uzy...	Utycie pro...	Czas proces...	Pamięć (K)	Wątki	Platfor...	Opis
TvUpdateInfo.exe	5884	Uruchomiony	adam.dept...	00	00:00:00		1	32 bity	TvUpdateInfo
tv_w32.exe	6712	Uruchomiony	SYSTEM	00	00:00:03		2	32 bity	TeamViewer
tv_w32.exe	7948	Uruchomiony	SYSTEM	00	00:00:03		2	32 bity	TeamViewer
tv_x64.exe	5940	Uruchomiony	SYSTEM	00	00:00:02		2	64 bity	TeamViewer
tv_x64.exe	7336	Uruchomiony	SYSTEM	00	00:00:02		2	64 bity	TeamViewer
unsecapp.exe	2664	Uruchomiony	SYSTEM	00	00:00:01		3	64 bity	Sink to receive
VGAuthService.exe	1872	Uruchomiony	SYSTEM	00	00:00:00		2	64 bity	VMware Guest
VikingSrvLuncher.exe	2304	Uruchomiony	adam.dept...	00	02:13:41	4 096 K	8	32 bity	VikingSrvLuncher
VikingSrvTcp.exe	6956	Uruchomiony	SYSTEM	00	00:36:31	6 680 K	18	32 bity	VikingSrvTcp
vikingsrvtcpjobs.exe	1740	Uruchomiony	adam.dept...	00	00:01:04	3 280 K	1	32 bity	vikingsrvtcpjobs
vikingsrvtcpsspy.exe	1196	Uruchomiony	adam.dept...	00	01:17:27	3 872 K	1	32 bity	vikingsrvtcpsspy
vikingsrvudp.exe	5876	Uruchomiony	adam.dept...	00	04:14:34	5 192 K	2	32 bity	vikingsrvudp

Cały ekosystem rozwiązania na serwerze zużywa zaledwie **2,3gb RAM**



10 wątków roboczych VFP w technologii COM zajmuje w pamięci zaledwie około **12 mgb RAM**.

infobcsrvmt.dll.exe	9084	Uruchomiony	SYSTEM	00	16 776 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	5932	Uruchomiony	SYSTEM	00	12 904 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	10740	Uruchomiony	SYSTEM	00	14 976 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	6668	Uruchomiony	SYSTEM	00	13 680 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	6344	Uruchomiony	SYSTEM	00	13 960 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	7200	Uruchomiony	SYSTEM	00	14 204 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	10536	Uruchomiony	SYSTEM	00	13 500 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	9584	Uruchomiony	SYSTEM	00	12 464 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll.exe	10152	Uruchomiony	SYSTEM	00	14 568 K	1	x86	infobcsrvmt.dll.exe	Niedostępny
infobcsrvmt.dll_foton.exe	10916	Uruchomiony	SYSTEM	00	5 340 K	1	x86	infobcsrvmt.dll_foton.exe	Niedostępny

Każdego dnia usługa Foxx-i Server odpowiada za zapisywanie, odczyt oraz edycję około **5 500** różnych dokumentów oraz **kilkudziesięciu tysięcy** paragonów fiskalnych z POS. Natomiast wszystkie instalacje Foxx-i Server w Polsce, w chwili obecnej zapisują dziennie ponad **pół miliona** paragonów. Foxx-i Server wykonuje średnio **350 tyś** operacji bazodanowych, z wydajnością 290 operacji na sekundę. Przy praktycznie zerowym USD nakładzie dla klienta za oprogramowanie bazodanowe.

*Adam Deptuła*