Budapesti Műszaki és Gazdaságtudományi Egyetem
Intelligens rendszerfelügyelet (BMEVIMIA370)

Házi feladat

3-C
Felügyeletre tervezés

Darvas Dániel
2011. április 21.

1 Bevezető

1.1 Feladatkiírás

A kiíráshoz mellékeljük egy egyszerű, C#-ban megvalósított kulcs-érték párokat tároló kiszolgáló és egy hozzá való kliens forráskódját (az alkalmazás 4.0-ás .NET Frameworkhöz készült az ingyenes Microsoft Visual C# 2010 Express Edition eszközzel).

Hogy jobban megértsük az alkalmazás működését, a forráskód tanulmányozásával készítsen egy UML modellt (tipikusan komponens vagy osztálydiagramokkal), ami ábrázolja a rendszer főbb részeit és azok kapcsolatait. A modell tartalmazza, hogy melyik komponens milyen adatot tárol, hogyan kommunikálnak egymással a komponenseink, tipikus esetben melyikből hány darab fut, stb.

Megfelelő érveléssel alátámasztva sorolja fel, hogy a kiszolgálónak milyen, a működésével kapcsolatos kvantitatív jellemzőket kellene rendszerfelügyeleti célokra elérhetővé tennie (pl. uptime)!

A metrikák definícióját valami jól áttekinthető formában adja meg (pl. táblázat), mely legalább a következő tulajdonságokat tartalmazza:

- metrika neve,
- metrika leírása,
- metrika számításának módja.

Ügyeljen arra, hogy a metrikák definícióját annyira formálisan és részletesen adja meg, hogy azok további pontosítás nélkül implementálhatóak legyenek (ez nem azt jelenti, hogy konkrét változóneveket kell megadni, csak annyit, hogy a számolásuk pontos mikéntjét leírjuk).

Legalább öt metrika definiálása kötelező.

WMI instrumentáció hozzáadásával és a szükséges egyéb módosításokkal tegye lehetővé, hogy a fentiek közül három metrika WMI segítségével, távolról lekérdezhető legyen!

Adja meg és indokolja, hogy milyen távoli rendszerfelügyeleti beavatkozásokat tart szükségesnek a kiszolgálóra (legalább három), és legalább egyet implementáljon ezek közül WMI metódusként úgy, hogy az pl. PowerShell-ből végrehajtatható legyen!

A megoldás dokumentációja a szokásos elemeken kívül (tervezési döntések, új osztályok/módosítások dokumentációja, tesztelés menete, környezet leírása, telepítés menete…) tartalmazza az 1. feladatban elkészített modell kiegészített változatát, illetve adjon példát a lekérdezésre és beavatkozásra (praktikusan PowerShell parancs formájában)!

1.2 Előkészítés

A Visual Studio projekt használatához telepítenem kellett a StyleCop és FxCop alkalmazásokat *(melyeket mellesleg nem ismertem, de nagyon hasznosnak találtam, középtávon mindkettőt szeretném a PetriDotNet projektben is használni)*.

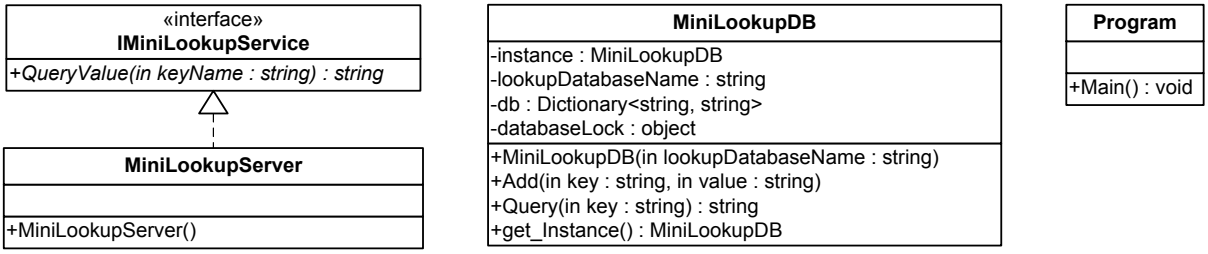
Más előkészítő lépésre nem volt szükségem.

2 Alkalmazás működésének értelmezése

Szerencsére az alkalmazás működésének megértése könnyű, hiszen bár a dokumentáció hiányzik, a forráskód igen egyszerű, intuitív és megfelelően kommentezett.

2.1 Szerver

A szerveralkalmazás lényeges komponensei az alábbi diagramon láthatók:



1. ábra A MiniLookupServer felépítése

A szerveralkalmazás belépési pontja a **Program** osztály statikus *Main* függvénye. Ez ellenőrzi a parancssori paramétereket (van legalább egy paraméter és ez helyes adatbázisnév: legfeljebb 8 karakterből áll és csak betűket tartalmaz). Amennyiben a kapott paraméter helyes, inicializálja az adatbázist (létrehoz egy új *MiniLookupDB* objektumot), illetve elindítja a *MiniLookup*-ot kiszolgáló WCF szolgáltatást. Ennek címe az alkalmazás beállításai közt beállított *baseAddress* címből és az adatbázis nevéből áll össze. Ez után egy végtelen ciklusban a következőket végzi:

- ha a beírt szöveg a q karakter, kilép a programból (előtte bezárja a WCF hosztot),
- ha a beírt szöveg tartalmaz vesszőt és ez nem az első vagy utolsó karakter, akkor felveszi az adatbázisba a megadott kulcs-érték párt,
- egyéb esetben nem csinál semmit a megadott szöveggel.

Az adatbázist a **MiniLookupDB** osztály valósítja meg. Ez egy Singleton minta alapján működik, az egyetlen példányt az *instance* mező tárolja, ezt az *Instance* tulajdonságon keresztül lehet elérni kívülről. Az osztályon belül egy *db* nevű string-párokat tartalmazó *Dictionary* tárolja a kulcs-érték párokat. Ehhez új elemet az *Add* függvénnyel lehet adni. Kulcs alapján értéket lekérdezni a *Query* függvénnyel lehetséges. Annak érdekében, hogy konkurenciaproblémák ne merüljenek fel, az osztályban van egy *databaseLock* objektum, amelyet zárolnak a kulcs-érték párok lekérdezését vagy módosítását végző függvények (a *Query* és az *Add*). Emellett a *lookupDatabaseName* stringben tárolásra kerül az adatbázis neve, mely értéket a konstruktor állít be.

A *MiniLookupDB* osztály távolról nem érhető el. Ehhez egy **IMiniLookupService** interfész került definiálásra, mely megadja, milyen metódus érhető el távolról. Jelen esetben egyetlen metódus, a *QueryValue* függvény, mely egy stringet kap paraméterül (a keresett kulcsot) és szintén egy stringgel tér vissza (a kulcshoz tartozó értékkel).

Ezt az interfészt a **MiniLookupServer** implementálja. Ebben a *QueryValue* függvény tulajdonképp a *MiniLookupDB* példányának a *Query* függvényét burkolja be, illetve egészíti ki hibakezeléssel. Hibás kulcs lekérdezése esetén, illetve ha a megadott kulcshoz nem tartozik érték, akkor megfelelően felparaméterezett *FaultException*-t dob.

2.2 Kliens

A kliensalkalmazás mindössze egyetlen Main függvényből áll. Ez először ellenőrzi, hogy megfelelő számú (1) paramétert kapott-e, majd a paraméterül kapott címen lévő MiniLookupServer-hez próbál kapcsolódni. Amennyiben sikerül, végtelen ciklusban lekérdezi a megadott kulcshoz tartozó érték párt. Ha ez sikeres, kiírja az értéket, ha sikertelen, hibajelzést ad. Q betű beírására kilép és lezárja a kapcsolatot.

3 Metrikák

3.1 A szerveralkalmazás lehetséges kvantitatív jellemzői

Az általam szükségesnek ítélt kvantitatív jellemzők a szerveralkalmazás esetén a következők:

Uptime

- metrika leírása: A szerver elindítása óta eltelt idő (másodpercben, kerekítve).
- indoklás: Ez a metrika jelzi, hogy mennyi ideje fut a szerver. Az alacsony uptime értékek szerver-újraindulást, kritikus hibákat jelezhetnek, ez pedig fontos információ.
- metrika számításának módja: A lekérdezés ideje és a szerver indításának ideje közt eltelt idő, másodpercben, egészre kerekítve.

Lekérdezések száma (QueryCount)

- metrika leírása: A szerver futása óta (távolról) lekérdezésre került értékek száma.
- indoklás: Ezzel a metrikával a szerver használata mérhető. Túl nagy, illetve túl kicsi kihasználtságot is jelezhet.
- metrika számításának módja: A szerver futása óta lekérdezésre került értékek száma, azaz az adott *MiniLookupServer* objektum *QueryValue* hívásainak száma.

Sikeres lekérdezések száma (SuccessfulQueryCount)

- metrika leírása: A szerver futása óta sikeresen (távolról) lekérdezésre került értékek száma.
- indoklás: Ezzel a metrikával a szerver effektív használata mérhető. Túl nagy, illetve túl kicsi kihasználtságot is jelezhet.
- metrika számításának módja: A szerver futása óta sikeresen lekérdezésre került értékek száma, azaz az adott *MiniLookupDB* objektum olyan *Query* hívásainak száma, amikor nem történt Exception.

Üres kulcsok lekérdezési száma (NullKeyQueryCount)

- metrika leírása: A szerver futása óta kezdeményezett lekérdezések száma, melyeknél a megadott kulcs null vagy nulla hosszúságú szöveg.
- indoklás: Ezzel a metrikával a kliens különféle hibái jelezhetők.
- metrika számításának módja: A szerver futása óta lekérdezésre került értékek száma, azaz az adott *MiniLookupDB* objektum *Query* hívásainak száma, amikor *KeyNotFoundException* kivétel került visszaadásra.

Nem létező kulcsok lekérdezési száma (NotExistingKeyQueryCount)

- metrika leírása: A szerver futása óta kezdeményezett lekérdezések száma, melyeknél a megadott kulcs nem található meg az adatbázisban.
- indoklás: Ezzel a metrikával a kliens különféle hibái jelezhetők.
- metrika számításának módja: A szerver futása óta lekérdezésre került értékek száma, azaz az adott *MiniLookupServer* objektum *QueryValue* hívásainak száma, amikor *KEY_NOT_FOUND* hibakód került visszaadásra.

A lekérdezések darabszámára vonatkozó metrikákból érdemes az elmúlt egy óra / egy nap / egyéb intervallumba eső lekérdezésszámot is elérhetővé tenni. Azaz hasznosak lehetnek a következő metrikák is:

- Lekérdezések száma az elmúlt órában
- Sikeres lekérdezések száma az elmúlt órában
- Üres kulcsok lekérdezési száma az elmúlt órában
- Nem létező kulcsok lekérdezési száma az elmúlt órában

Ezek részletesebb ismertetését nem tartom szükségesnek.

Tárolt kulcs-érték párok száma (ItemCount)

- metrika leírása: A szerveralkalmazás által tárolt kulcs-érték párok száma.
- indoklás: Ezzel a metrikával a szerver által tárolt adatmennyiség figyelhető meg. Kis értékek esetén az üzemeltetés esetleg felesleges, míg nagy érték esetén megfontolandó más tárolómegoldás használata.

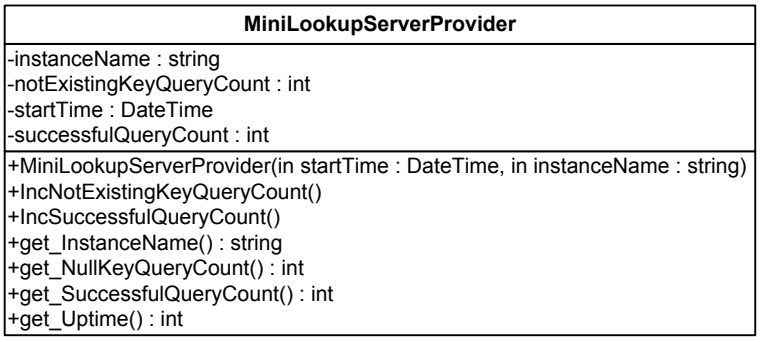
• metrika számításának módja: A tárolt kulcs-érték párok száma, azaz a konkrét *MiniLookupDB* példány db mezőjében tárolt bejegyzések száma.
A szerver aktuális processzorterhelése (CurrentCPUUsage) • metrika leírása: A szerveralkalmazás aktuális processzorterhelése (százalékban). • indoklás: Ezzel a metrikával a szerver erőforrásigénye figyelhető meg. • metrika számításának módja: A metrika számítását az operációs rendszer végzi, a programnak csak le kell kérdeznie ezt.

Ezek mellett természetesen más metrikák definiálása is értelmes lehet, de ez függ a konkrét felhasználási módtól.

3.2 Távolról lekérdezhető metrikák implementálása

A fenti metrikák közül az uptime, valamint a sikeres lekérdezések számának és az nem létező kulcsok lekérdezési számának lekérdezhetőségét implementáltam.

Ehhez létrehoztam a *MiniLookupServerProvider* osztályt. Ez tartalmazza a lekérdezéshez szükséges tulajdonságokat (C# property-ket), illetve a lekérdezéshez szükséges adatokat is. Az osztály mezőiben tárolom az aktuális példány nevét (*instanceName*), a példány elindulásának időpontját (*startTime*), valamint az üres és sikeres lekérdezések eddigi számát (*nullKeyQueryCount* és *successfulQueryCount*). Ez utóbbiak azért kerültek ide, hogy minél kevesebb kóddal kelljen kiegészíteni az eredeti osztályokat. A lekérdezések számának növelésére létrehoztam egy *IncNotExistingKeyQueryCount* és egy *IncSuccessfulQueryCount* metódust, melyek a megfelelő számlálót növelik meg eggyel. Emellett létrehoztam az *InstanceName*, *NotExistingKeyQueryCount*, *SuccessfulQueryCount* és *Uptime* tulajdonságokat, melyek az azonos nevű (de kisbetűvel kezdődő) mezők publikus elérhetőségét szolgálják. Ezek közül az első egy kulcsérték, a többi pedig lekérdezhető metrika, ennek megfelelően láttam el *ManagementKey* vagy *ManagementProbe* attribútummal.



2. ábra A MiniLookupServerProvider osztály

A MiniLookupServerProvider osztály forrráskódja a következő:

```
[ManagementEntity(Name="MiniLookupServer"),
     ManagementQualifier("Description", Value = "Provider for MiniLookup server.")
]
    public class MiniLookupServerProvider
    {
        // A szerver indulásának ideje.
        private DateTime startTime;

        // A szerveralkalmazás példányának neve.
        private string instanceName;
```

```
// A sikeres lekérdezések száma.
private int successfulQueryCount = 0;

// Az üres kulcsok lekérdezésének száma.
private int notExistingKeyQueryCount = 0;

// Konstruktor a MiniLookupServerProvider-hez
internal MiniLookupServerProvider(DateTime startTime, string instanceName)
{
    this.startTime = startTime;
    this.instanceName = instanceName;
}


[ManagementKey]
public string InstanceName
{
    get
    {
        return instanceName;
    }
}

[ManagementProbe]
public long Uptime
{
    get
    {
        return (long)Math.Round((DateTime.Now - startTime).TotalSeconds);
    }
}

internal void IncSuccessfulQueryCount()
{
    successfulQueryCount++;
}

[ManagementProbe]
public int SuccessfulQueryCount
{
    get
    {
        return successfulQueryCount;
    }
}

internal void IncNotExistingKeyQueryCount()
{
   notExistingKeyQueryCount++;
}

[ManagementProbe]
public int NotExistingKeyQueryCount
{
    get
    {
        return notExistingKeyQueryCount;
    }
```

```
        }
}
```

Természetesen az eredeti programot is ki kellett egészíteni annak érdekében, hogy lekérdezhetők legyenek a fenti metrikák.

Egyrészt a MiniLookupDB-t kiegészítettem egy Provider publikus, írható-olvasható tulajdonsággal.

```
/// <summary>
/// WMI provider.
/// </summary>
public MiniLookupServerProvider Provider {get; set;}
```

A Main függvény inicializációs részében létrehozásra és publikálásra került a provider. Ez sikeres létrehozás esetén át lett adva a MiniLookupDB-nek.

```
 // Initialization of WMI provider.
MiniLookupServerProvider provider =
new MiniLookupServerProvider(DateTime.Now, args[0]);
try
{
      InstrumentationManager.Publish(provider);
}
catch
{
      provider = null;
      Console.WriteLine("WARNING: The publication of WMI provider was unsuccessful
.");
}

lookupDB = new MiniLookupDB(args[0]);
lookupDB.Provider = provider;
```

Emellett a MiniLookupDB osztály Query függvényét kiegészítettem a dőlten szedett sorokkal:

```
internal string Query(string key)
{
      lock (this.databaseLock)
      {
            if (!this.db.ContainsKey(key))
            {
                  if (Provider != null)
                        Provider.IncNotExistingKeyQueryCount();
                  throw new KeyNotFoundException();
            }

            if (Provider != null)
                  Provider.IncSuccessfulQueryCount();
            return this.db[key];
      }
}
```

Ezek a sorok amennyiben van beállított provider, a megfelelő számlálók értékeit léptetik.

3.3 Üzembe helyezés, tesztelés

A teszteléshez először telepítenem kellett a WMI szolgáltatót. Ehhez az installutil.exe-t használtam az alábbi paraméterezéssel, rendszergazda jogú parancssorban:

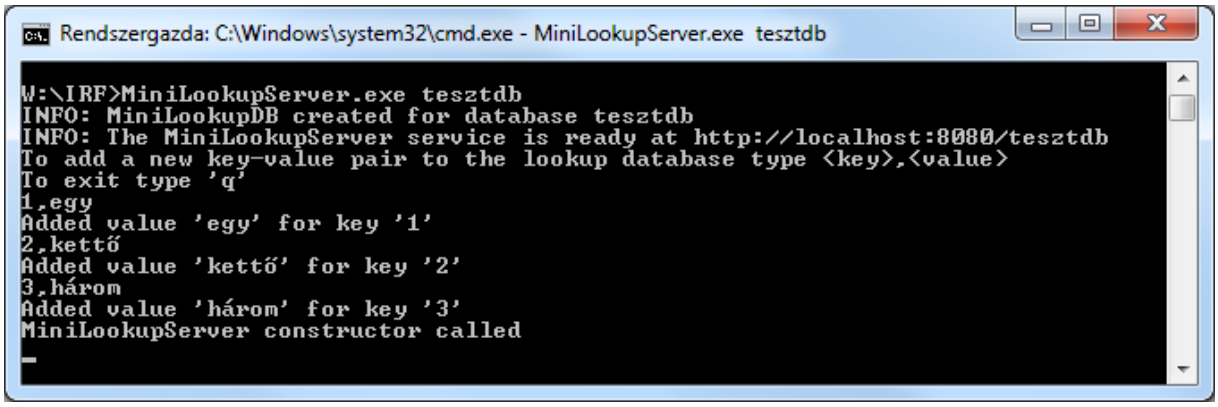
```
c:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe MiniLookupServer.exe
```

A futtatás sikeres volt ("The transacted install has completed.").

Ez után elindítottam egy szerverpéldányt, illetve egy klienspéldányt. A szerverpéldány elindítását rendszergazda jogú parancssorban végeztem.

A szerver indításához használt parancs:

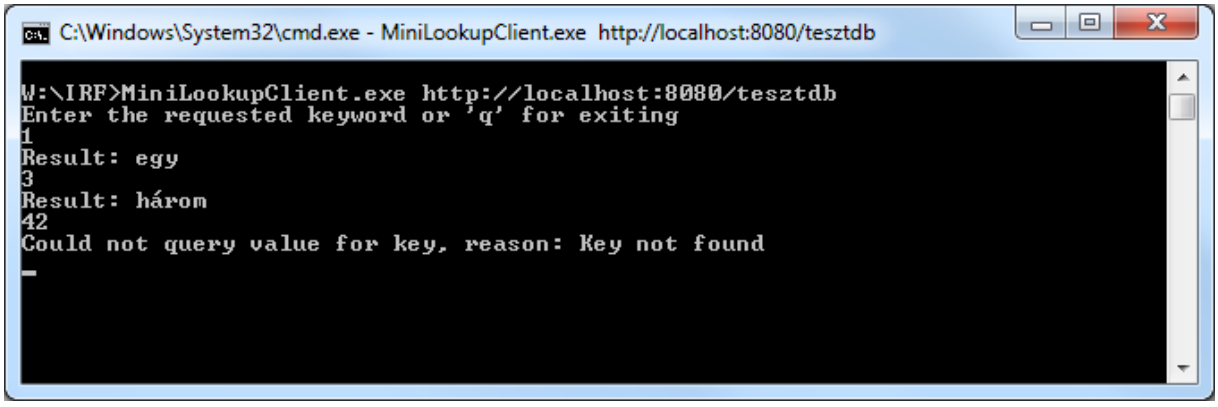
```
MiniLookupServer.exe tesztdb
```



3. ábra A teszteléshez használt szerverpéldány

A kliens indításához használt parancs:

```
MiniLookupClient.exe http://localhost:8080/tesztdb
```



4. ábra A teszteléshez használt klienspéldány

A parancssori kimeneteken látható, hogy a kapcsolat felépült és lehetséges lekérdezéseket is végezni.

Ez után teszteltem a metrikák lekérését WMI-n keresztül, PowerShellből, az alábbi paranccsal:

```
Get-WmiObject MiniLookupServer -Namespace root\irf
```

Ennek eredménye látható a következő ábrán:

```
PS W:\IRF> Get-WmiObject MiniLookupServer -Namespace root\irf


__GENUS                 : 2
__CLASS                 : MiniLookupServer
__SUPERCLASS            :
__DYNASTY               : MiniLookupServer
__RELPATH               : MiniLookupServer.InstanceName="tesztdb"
__PROPERTY_COUNT        : 4
__DERIVATION            : {}
__SERVER                : KRTEK
__NAMESPACE             : root\irf
__PATH                  : \\KRTEK\root\irf:MiniLookupServer.InstanceName="tesztdb"
InstanceName            : tesztdb
NotExistingKeyQueryCount : 1
SuccessfulQueryCount    : 2
Uptime                  : 15
```

5. ábra Egy WMI-lekérdezés eredménye

Látható, hogy a *NotExistingKeyQueryCount* értéke 1 (a 42 lekérdezése miatt), a *SuccessfulQueryCount* értéke 2 (az 1 és a 3 lekérdezése miatt). Emellett leolvasható az is az Uptime értékéről, hogy a szerveralkalmazás 15 másodperce fut.

Kipróbáltam azt az esetet is, amikor a szerveralkalmazás nem fut. Ekkor is az elvárt működést tapasztaltam, azaz a *Get-WmiObject* cmdlet nem adott vissza eredményt.

```
PS W:\IRF> Get-WmiObject MiniLookupServer -Namespace root\irf
PS W:\IRF>
```

6. ábra Egy WMI-lekérdezés eredménye leállított szerveralkalmazás esetén

4 Távoli beavatkozási lehetőségek

4.1 Lehetséges távoli beavatkozási lehetőségek

Az alábbi távoli rendszerfelügyeleti beavatkozások lehetnek szükségesek (a felhasználás módjától függően):

- Szerveralkalmazás leállítása
- Új kulcs-érték pár felvétele
- Kulcs-érték párok törlése

Az utolsó két beavatkozási lehetőség nem klasszikus értelemben vett rendszerfelügyeleti beavatkozási lehetőség, azonban elképzelhető olyan felhasználási terület, ahol ennek értelme van, a program igen általánosan felhasználható.

4.2 Távoli beavatkozási lehetőségek implementálása

Az előzőekben vázolt beavatkozási lehetőségek közül a kulcs-érték párok felvitelét implementáltam. Ehhez kiegészítettem a MiniLookupServerProvider osztályt egy AddItem metódussal, amely egy stringet vár bemenő paraméterként, olyan formátumban, ahogyan a konzolon is meg kell adni a kulcs-érték párost.

Az implementált metódus az alábbi:

```
[ManagementTask]
public void AddItem(string input)
{
	string[] inputs = input.Split(',');

	if (inputs[0].Length == 0 || inputs[1].Length == 0 || inputs.Length != 2)
	{
```

```
            throw new ArgumentException("Argument error: zero-
length key or value.");
      }

      string key = inputs[0];
      string value = inputs[1];

      MiniLookupDB.Instance.Add(key, value);
}
```

A kiegészítés után a *MiniLookupServerProvider* osztály az alábbi ábrán látható lett.

MiniLookupServerProvider
-instanceName : string -notExistingKeyQueryCount : int -startTime : DateTime -successfulQueryCount : int
+MiniLookupServerProvider(in startTime : DateTime, in instanceName : string) +AddItem(in item : string) +IncNotExistingKeyQueryCount() +IncSuccessfulQueryCount() +get_InstanceName() : string +get_NullKeyQueryCount() : int +get_SuccessfulQueryCount() : int +get_Uptime() : int

7. ábra MiniLookupServerProvider kiegészítve rendszerfelügyeleti beavatkozással

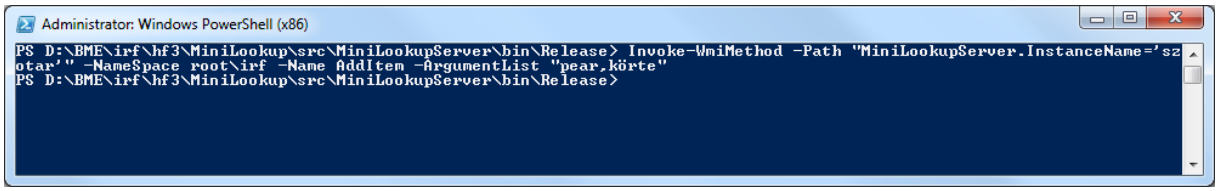
4.3 Tesztelés

A tesztelést PowerShellből végeztem, az alábbi módszerrel:

```
Invoke-WmiMethod -Path "MiniLookupServer.InstanceName='<instance name>'" -
NameSpace root\irf -Name AddItem -ArgumentList "<key>,<value>"
```

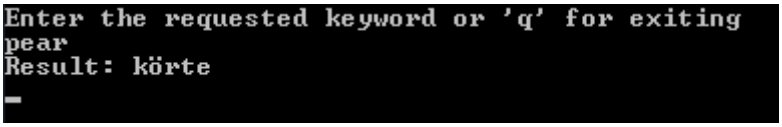
Például:

```
Invoke-WmiMethod -Path "MiniLookupServer.InstanceName='szotar'" -NameSpace
root\irf -Name AddItem -ArgumentList "pear,körte"
```



8. ábra Az AddItem metódus meghívása távolról

Ez után a kliensben a pear kulcshoz tartozó érték lekérdezése sikeres volt:



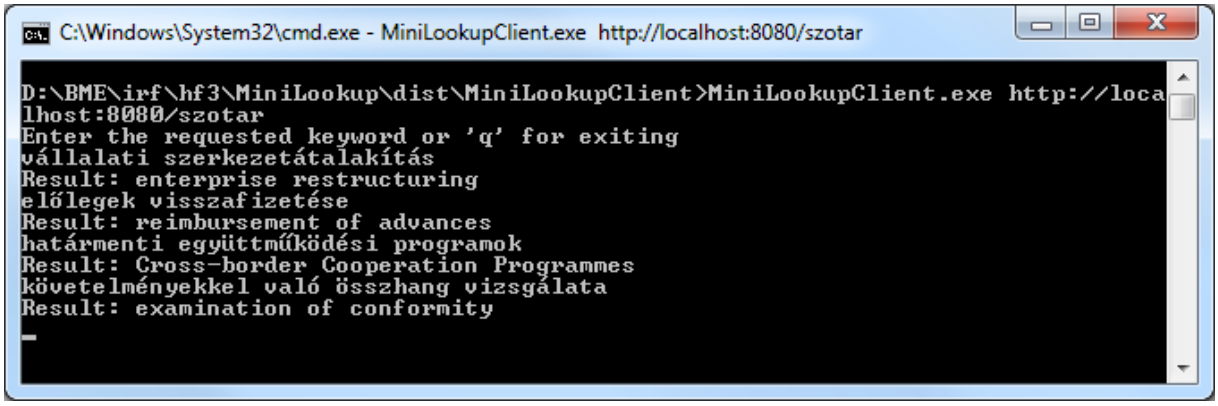
9. ábra Az AddItem hívás eredményének tesztelése

Amennyiben hibás paramétert adok meg, például már létező kulcshoz próbálok új értéket felvenni, a felhasználó hibát fog kapni. Ez szándékos, direkt nem kezeltem le az esetleges kulcsütközést, mivel a csendes hiba véleményem szerint sokkal veszélyesebb.

```
PS D:\BME\irf\hf3\MiniLookup\src\MiniLookupServer\bin\Release> Invoke-WmiMethod -Path "MiniLookupServer.InstanceName='sz
otar'" -NameSpace root\irf -Name AddItem -ArgumentList "pear,banán"
Invoke-WmiMethod : Invalid parameter
At line:1 char:17
+ Invoke-WmiMethod <<<<  -Path "MiniLookupServer.InstanceName='szotar'" -NameSpace root\irf -Name AddItem -ArgumentList
 "pear,banán"
    + CategoryInfo          : InvalidOperation: (:) [Invoke-WmiMethod], ManagementException
    + FullyQualifiedErrorId : InvokeWMIManagementException,Microsoft.PowerShell.Commands.InvokeWmiMethod
```

10. ábra Az AddItem metódus hibás meghívása távolról

Kipróbáltam az AddItem hívást nagyobb adathalmazzal is. Ehhez a MEK-ból letöltöttem a Magyar-angol EU-szótárat[1], és ennek a kb. 800 kifejezéses szószedetét töltöttem fel. Az ehhez használt script megtalálható a beadott megoldásban *szotar-upload.ps1* néven. Így már olyan, nap mint nap használt kifejezések fordítása lehetséges ezzel a programmal, mint a vállalati szerkezetátalakítás vagy a követelményekkel való összhang vizsgálata.



11. ábra Lekérdezés tesztelése az EU-szótárral feltöltött példánnyal

5 Tesztelés külön számítógépen futó komponensek esetén

Teszteltem az alkalmazást úgy is, hogy a kiszolgáló komponens, illetve a kliens külön (virtuális) számítógépen fut. A WMI-lekérdezéseket is azon a gépen végeztem, amelyen a kliens futott.

A szerveralkalmazást futtató virtuális gép a 192.168.152.134 IP-címen futott. A klienssel az alábbi paranccsal csatlakoztam a szerverhez:

```
MiniLookupClient.exe http://192.168.152.134:8080/szotar
```

Ez elsőre sikertelen volt. A szerveralkalmazást futtató gépen a 8080-as portot kinyitva viszont már működött a lekérdezés hálózaton keresztül. Emellett a távoli WMI-lekédezésekhez engedélyeztem a DCOM-hoz szükséges portokat is a következő paranccsal:

```
netsh firewall set service RemoteAdmin enable
```

A távoli WMI-lekérdezésekhez a korábbiakban használt cmdletet ki kellett a ComputerName paraméterrel és a kapcsolódáshoz használt felhasználónévvel egészítenem, így a lekérdezéshez használt parancs így alakult:

```
Get-WmiObject MiniLookupServer -Namespace root/irf -ComputerName 192.168.152.134 -
Credential Administrator
```

Eredménye az alábbi lett:

[1] http://mek.niif.hu/00000/00081/00081.htm

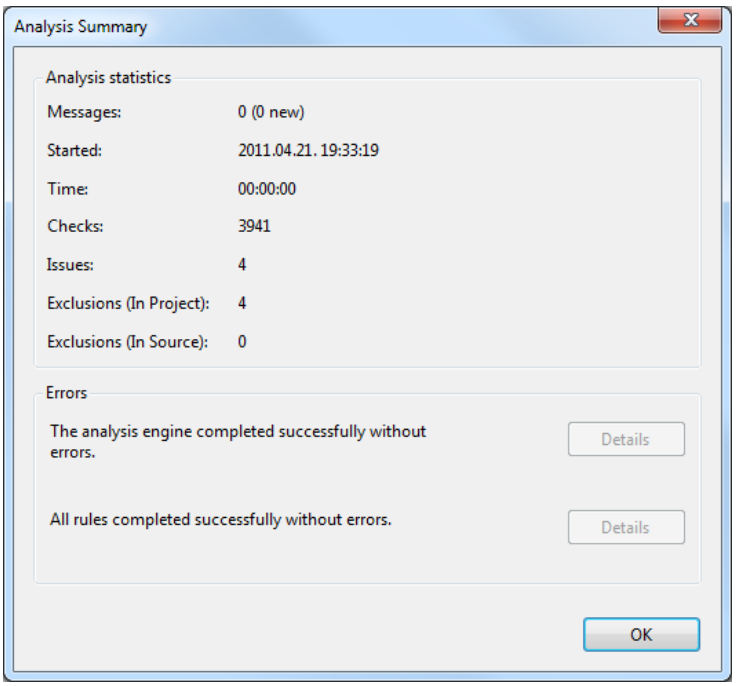
```
PS W:\IRF> Get-WmiObject MiniLookupServer -Namespace root/irf -ComputerName 192.168.152.134 -Credential Administrator


__GENUS                : 2
__CLASS                : MiniLookupServer
__SUPERCLASS           :
__DYNASTY              : MiniLookupServer
__RELPATH              : MiniLookupServer.InstanceName="szotar"
__PROPERTY_COUNT       : 4
__DERIVATION           : {}
__SERVER               : IRFSERVER
__NAMESPACE            : root\irf
__PATH                 : \\IRFSERVER\root\irf:MiniLookupServer.InstanceName="szotar"
InstanceName           : szotar
NullKeyQueryCount      : 0
SuccessfulQueryCount   : 1
Uptime                 : 64
```

12. ábra Távoli WMI-lekérdezés

Azaz sikeres volt a távoli gépről a WMI-lekérdezés.

6 Egyéb megjegyzések

A munkámat ellenőriztem az FxCop eszközzel. Az így feltárt hibák közül kijavítottam azokat, melyeket én vétettem, de az "Excluded In Project"-be sorolt problémák javításával nem foglalkoztam. A leadott változatra az eszköz már nem jelez aktív státuszú hibát.



13. ábra Az FxCop analízisének eredménye