

# Singularity

System operacyjny napisany w .NET  
(ogólnie rzecz biorąc)

# Singularity? Co to jest?

- ◉ System operacyjny napisany przez Microsoft Research
- ◉ Co by było, gdyby zacząć od nowa?
- ◉ A gdyby się nie skupiać na szybkości, ale na „dependability”?
- ◉ Napisany w kodzie zarządzanym
- ◉ Udostępniony kod źródłowy (4 marca 2008)
- ◉ RDK 2.0 (listopad 2008)
- ◉ <http://singularity.codeplex.com>

# Dlaczego obecne systemy są do kitu?

- ◉ Nowe zagrożenia (kod webowy)
- ◉ Nowe metody wykorzystania (mikrofalówka)
- ◉ Nowy sprzęt (GPGPU)
- ◉ Kompatybilność wsteczna (miecz obusieczny)

# Czym się różni?

- SIP = Software Isolated Process
- Bezpieczeństwo, bezpieczeństwo, bezpieczeństwo
- Metadane to podstawa
- CLR w jądrze
- Kod zarządzany – MSIL i „bare metal”
- Kanały, kontrakty, sterła wymiany

# SIP

- ◉ Procesy w Singularity są zamknięte w izolowanych klatkach: SIP-ach
- ◉ SIP-y to przestrzenie obiektów, nie adresów
- ◉ SIP nie może dynamicznie generować ani ładować kodu
- ◉ SIP-y nie polegają na sprzętowych mechanizmach izolacji – tylko software
- ◉ Komunikacja pomiędzy SIP-ami dostępna jest tylko poprzez dwukierunkowe, silnie typowane kanały (*channels*)
- ◉ Każdy proces, sterownik, biblioteka, rozszerzenie ma własnego SIP-a

# Bezpieczeństwo

- Izolacja
- Typy, typy, typy!
- Kod może być zweryfikowany (*verified*) lub zaufany (*trusted*)
- Weryfikacja jest przeprowadzana przez kompilator
- Większość kodu jest weryfikowalna, oprócz części napisanych w C, C++ i assemblerze (ok. 5%)
- Cały inny kod jest pisany w bezpiecznych typowo językach, kompilowanych do kodu pośredniego MSIL/CIL
- Ufamy, że Bartok kompiluje kod sprawdzając, czy jest bezpieczny
- Plan użycia typowanego assemblera – gdyby taki istniał :-)

# Kanały (*channels*) i kontrakty (*contracts*)

- Kanały: dwukierunkowe, typowane połączenia pomiędzy procesami
- Kontrakty: definicje wysyłanych i odbieranych przez procesy wiadomości i ich typy
- + Endpointy – wygodne wysyłanie i odbieranie wiadomości

```
contract C1
{
    in message Request(int x) requires x>0;
    out message Reply(int y);
    out message Error();

    state Start: Request?
        -> (Reply! or Error!)
        -> Start;
}
```

# Kod w liczbach

- ok. 360 tys. SLOC
- Wg standardowego modelu COCOMO – 3 lata pracy 30 programistów
- 13 milionów dolarów ;-)
  
- Linux – ok. 12 mln SLOC
- Windows Server 2003 – ok. 50 mln SLOC

# Wydajność

	Singularity	FreeBSD	Linux	Windows
Read cycle counter	8	6	6	2
ABI call	87	878	437	627
Create and start process	300 000	1 032 000	719 000	5 376 000

# Wydajność

	Singularity	FreeBSD	Linux	Windows
C		1 200	1 416	644
C (statyczny)		232	664	544
C++		2 148	2 532	804
C++ (statyczny)		704	1 216	572
Sing#/C#	408			4 116

Rozmiar Virtual Address Space dla Hello World (w KB)

# Czas na pokaz!

- ◉ Budowanie systemu
- ◉ Uruchamianie systemu na maszynie wirtualnej
- ◉ Debugger jądra

# Programowanie Singularity

- Zaraz napiszemy własną aplikację :-)
- I dowiemy się dlaczego nie jest tak różowo ;-)

# Zastosowania

- RDK tylko i wyłącznie do celów badawczych i edukacyjnych
- Ale wg. User Application Group: PVR, Audio PC, Car PC, NAS
- Virtual PC na Singularity
- Wewnętrzne wdrożenia Microsoftu

# Singularity a nowe Windows

- Microsoft Midori
- Windows 9 czy rezygnacja z nazwy Windows?
- Windows cięży kompatybilność wsteczna
- Windows cięży parę innych rzeczy

