CosmoS

Cosmos-B

OVERVIEW, DeSIGN GOALS AND MOTIVATION

*Cosmos-B is a managed code operation system designed to run on the Cosmos kit. This document describes the reasons and priorities for building the OS. .*

DESIGN GOALS AND MOTIVATION

Ben KloosterMan

# ISSUES WITH CURRENT OS

Biggest issues with Windows ( and most OS)

* Crashes often caused by faulty drivers
* Security and Viruses especially Trojans and Keyloggers
* Death by paging
* A large app hogging resources especially disk and slowing other tasks down waiting for those resources.
* Bloated applications
* Unstable applications
* Inconsistent logging by systems, applications etc
* Corrupted Registry
* Lack of change / difficult to change
* Lack of embedded style reliability.
* Difficult to scale to small devices

The design will relate back to solving these issues.

# 1.Motivation

The ComsosB project is an OS that will attempt to make a production ready OS based on the promise of an OS based on a strongly typed language. A “Managed OS” is an OS that is based on a strongly typed language that uses Garbage Collection and uses languages features to improve security.

A managed OS promises to make improvements in performance, reliability and security compared to other common designs for the last 30 years. While software itself has changed greatly OS are pretty much written the same way they were 30 years ago, few of the improvements in software development are applied to Operating System development. We intend to use modern software techniques such as Object Orientation, Test Driven Development and Service Orientation within the Operating System.

It is worth stating what we are doing is not really new, managed application servers are common in the Java and .NET fields. The biggest weakness with these application servers is security which is primarily due to limitations in the underlying operating system. We are just extending the application server to be rather than use the OS.

The biggest issue with a managed OS is the requirement that all applications be managed applications, however with the advent of .NET and Java ( and to a lesser extent PHP , Python and Perl) we can maintain backward compatibility for a significant amount of applications.

Cosmos-B is a single-address-space operating system which can avoid the overhead of hardware-MMU faults and lookups and copies between protection domains because it uses a typesafe and verifiable execution core.

This also makes the system simpler and more flexible and uses stronger OO techniques favoring design/maintainability and advanced algorithms and not tight static pointer rich code.

# 2. PRIORITIES

**2.1 Simplicity and Maintainability**

The most important priority is for the Operating system to be easy to change and for the code to be read easily and understood by a novice. This will allow the OS to better handle changes as they are required.

To aid in this we will use the Cosmos kit which allows the Operating system to be built, debugged and launched directly from Visual Studio. By treating it as a normal application developers will be able to use all their existing experience.

Trusted code in traditional systems becomes reliable through maturity and testing . To increase reliability of the TCB itself the system will exhibit strong OO code quality implemented and tested through TDD. (Except for a few assembler “Plugs”) , which will allow developers to maintain confidence in the code base , allow changes and embed learned issues into the test cases for future reference.

The code itself will be OO code with little global /static data. Even services like the memory manager will be instance objects which can be changed as needed.

**2.2 Reliability**

It is frequently said that the greatest opportunity for OS advancement is in system reliability. The system will aim to detect problems for applications as early in the development process as possible, just as is the software development cycle early discovery makes the problem easier to deal with and results in a more reliable system. Eg. an application that is nor verifiably type and memory space will not even install.

Eg

Design -> Compile -> Test ->Install ->Load ->Run ->Post Mortem

**2.2.1 Device driver Reliability**

A traditional problem for OS is device drivers. For Comsos-B device driver’s are normal TSIA managed by the OS. They are also self describing so the OS can managed the drivers ,what hardware they apply to and avoid conflicts Device drivers do not touch the hardware directly all calls are done via the HAL ( which calls machine specific methods) which limits what the driver can do according to the agreed MetaData..

**2.3 Security**

The system adheres to both POLP /POLA as much as possible. POLA states that every module (such as a process, a user or a program on the basis of the layer we are considering) must be able to access only such information and resources that are necessary to its legitimate purpose. To adhere to this Cosmos uses a Capability Object system using references as the Capability. These references are stored and persisted in a key ring for each STP ( and reloaded when the STP loads) . In a well written program only a small part of the program will be able to use a certain privilege eg write to a output file. The rest of program will not have that ability.

**2.4 Scalability and Performance**

Modern Operating Systems were designed according to the hardware of 80-90 and hence were not well designed to handle modern multi core ,multi Gigabyte memory hardware. The high count of multiple cores (which is rapidly increasing), the slow increase of memory speed and the use of powerful non uniform CPU cores (like GPUs) are not well used. Eg in 2010 6 Core CPUs with HT will be released on quad board that’s a machine running 48 Hardware threads , 2011 consumer CPUs will have 4-8 cores or 8-16 Hardware threads. 64 bit support is also poor in a lot of modern Operating Systems with changes often limited to the memory management system.

The significant increase in cores work far better using low over head IPC [Citation]. Not HW protected OS can provide this much lower latencies and hence improve multi core performance.

The goals of the system are to be efficiently scalable on modern hardware from 32 Meg 400Mhz arm units with no HDD to servers with 100 Gig of Memory and are capable of running over 100 hardware threads on non homogenous CPUs .

Special attention is to be given to supporting the nature of the internet and efficient Asynch communication.

**2.5 Backward compatibility**

The system will support Mono and NET application and it will later support Java.

# 3.Key Design points /Features

The following key design points combine to produce an OS that satisfies our motivation that is flexible, maintainable, secure and more reliable these design points/features are:

* **A type safe versioned Asynchronous messaging system used for all IPC and kernel API / ABI calls.**
* **More infrequent IPC / Kernel calls by using references or sending larger structures.**
* **A kernelless structure where each app is equal hence there is little reason to trust callers.**
* **Strong software based application isolation architecture removing the need for expensive MMU based protection**
* **A OS assisted breakdown of applications to support better execution on multi core hardware and greater reuse of shared components.**
* **Ubiquitous metadata describing code and data enabling the OS to understand the requirements better.**
* **A POLP /POLA Object /Capability security model**
* **Orthogonally Persistent Applications**
* **Gradual failure of application and automatic restart for failing parts**
* **System uses strong OO code with little or no global data and is hence very modular.**

# 4.DEFINITIONS

Terms that shall not be used.

* Process – overloaded
* AppDomain – Nice term but too .NET specific.
* SIP – We don’t use process
* Client – Overloaded
* Server – Overloaded.
* DLL – Unmanaged ( use Assembly)
* MSIL – replaced by CIL.

Terms that should used with caution

* GC – use rarely Avoid GC use the term Allocator or Collector
* Application – over used
* Kernel – meaningless. There is only trusted and un-trusted code. Trusted assemblies run inside user applications. Use only to compare to kernel based OS.

Terms that should be used

* Trusted code (or trusted code base). Code that the OS trusts ( ie code we ship) . This code uses pointers and IO instructions and we know works either through verification, unit tests, maturity or a combination of factors.
* User code User code ie untrusted , includes file systems ,device drivers etc.
* Async-Client – when referring to a STP IPC caller.
* Async-Server – when referring to a STP IPC server and message pump
* Object Capability (OC) or Capability – A security key which accesses the object its basically a reference but can be persisted. The term often indicates some trusted privilege eg Hal Capability. Allows communication with the HAL.
* Managed Operating System – A type safe , Memory safe operating system that uses extensive Metadata and Garbage Collection to manage apps
* SingleThreadProcess(STP) – A single thread which executes over an object space, has an IPC, Scheduling , Sub Thread Process , Message pump and a reference to an SIA for Allocation and object creation capabilities as well as Capability storage.
* Named STP A Named STP is predefined in an SIA meta data file. Since they are named they are discoverable. Named STPs can act as services for messaging unnamed STPs can only be clients. There is always the default Based on the SIA Name + \_Main .
* Dynamic Loader – Loads Executables and Assemblies , verifies and then Assembly links them.
* Software Isolated Application (SIA) – A collection of 1 or more Thread Processes and access to a Collector Domain. Has capabilities to Code Assemblies. Its isolated because new Assemblies (and hence code) will not be allowed after it has loaded and existing code is verified to ensure its consistent with the Collector Domain and allowable Assemblies.

It has no threads. May contain sub Isolated Applications. It has no role in terms of execution, IPC and scheduling but is responsible for holding the references to verified code , principal security and memory management as well as persistence.

* Collector Domain – A collector manages memory. All code sharing a Collector is in the Collector domain ( and it could be everything in the system)
* Assembly /Trusted Assembly - A code library.
* Assembly Linking – The process where external calls are patched to the correct memory location when it is loaded.
* Message Pipe - Refers to an IPC pipe used for default communication in the system ie the API/ABI. It stores the sender, receiver and is point to point also does the security check at the start.
* CIL is the intermediate representation used for all code.
* Batch Application Context – Context used to pass information from one application to the next. ( like standard output but for any type and always strongly typed)

# 5.DOCUMENT NOTES

NOTE THIS DOCUMENT IS NOT A PAPER BUT A COHERENT OS FROM WHICH TO BEGIN CODING. The goal is to avoid parts that won’t work well together and explore a new OS with concepts from the ground up.

### Sample Code

Any sample code contained here is for illustrative purposes only and is not by any means final. Actual implementation likely will vary.

# 6.SYNERGY WITH OTHER DEVELOPMENT

The increased use in recent times of GC applications and more recently highly parallel languages based on Immutable types (Caml/F#) provides a very strong synergy with CosmosB.

# 7.Documents

Document for Cosmos B

* 0 – Overview, Design Goals and Motivation
* 1-Overall architecture
* 2-STP’s ( Process Model)
* 3-IPC & Shared Memory
* 4-Memory Management
* 5- Garbage Collector
* 6- Security
* 7-UI
* 8- Scheduler
* 9- Device Driver Model
* 10- Memory Manager
* 11 System Libs