INFERNO OPERATING SYSTEM

Olugbenga O. Fadiya

Loyola University Chicago

Dept. of Computer Science

ABSTRACT

The purpose of this document is to give a brief introduction and overview of the operating system known as Inferno. An operating system (OS) is an integral part of a computer system which not only provides an image of the machine to a program but also deals with the management of resources and provision of common services to application software. In other words, an Operating System ensures that different programs are able to interact with one another with interference. Operating systems are classified into different categories one of which is known as distributed operating system and where the OS Inferno belongs.

This paper is not intended to cover the depth, complexity and implementation of this operating system in an/any organization but focus herein will be on the general review of the components that constitutes an OS, specifically structure, components, algorithms involved in an operating system and user space component interaction with an operating system such as Inferno.

Inferno Operating System

Inferno is an operating system developed and first released in 1996 by the Computing Science Research Center of Bell Labs for creating and supporting distributed services. Though originally designed as a commercial product intended for the marketplace by Bell labs, it is now an open source software managed by Vita Nuova Holdings with the latest release in 2007. Inferno as an Operating System was derived from another of Bell Labs operating system known as Plan 9, though fundamentally different in various ways, it does shares basic principles such as - resources as files, Namespaces and Standard communication protocol with Plan 9 **(http://en.wikipedia.org/wiki/inferno_operating_system - emphasis added).**

Inferno is by all accounts an operating system which possesses interesting features. While the Inferno OS has the capability to run directly on any compatible hardware architecture, it can also function as an application which provides a virtual operating system over other platforms utilizing its virtualization capability. The portability and versatility of this Operating System also cuts across several dimensions which includes but not limited to its' ability to run on major hardware architectures, portability across different environments, distributed system design, minimal hardware requirements and dynamic adaptability for applications (**The Inferno Operating System, Doward, etc. 1997).**

The scope of the Inferno Operating System is a very expansive domain and beyond what could be potentially covered here; this paper will focus on the general overview of components that constitutes an OS and its implementation or lack thereof in Inferno, specifically structure, components, algorithms and user space component interaction with an operating system such as Inferno. Given the fact that it will be nearly impossible to summarize all the relative internal and important components of an OS within this paper, for clarity sake, there will be an attempt to achieve this goal by exploring and analyzing the design principles of Inferno as a distributed operating system, with this, we can gain an insight into the overall functionality of the Inferno from the perspective of a distributed operating system. We will then examine specific areas of this OS relative to critical components which have

already been alluded to above especially its' performance in these areas.

It will be a daunting task to completely comprehend and appreciate the many features an Operating System such as Inferno offers without first familiarizing ourselves with the concept of a Distributed Operating System. This concept can simply be explained as one where transparency is the key driving factor, in other words - it is "a system which appears to the user as an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs). The aim of a distributed operating system is to offer a single system image by utilizing transparency and fault-tolerance ***(Distributed Operating Systems – Tanenbaum and Van Renesse, 2002).***

In the context of a distributing operating system, files and programs are automatically processed and handled efficiently by the operating system though on multiple processors but which to a user seems like it is all been handled by a traditional uniprocessor. If we have defined a distributed operating system as that whose key concept is providing transparency to all resources through encapsulation, therefore the many features and strengths of the Inferno as a distributed operating system cannot be over – emphasized. Distributed operating systems in contrast to traditional systems, clearly its' biggest advantage is the ability to allow applications to run on multiple processes, needless to mention this advantage and distribution differs in so many ways and thus requires very complex processor scheduling algorithms to optimize the amount of parallelism **(Operating System, Design and Implementation** - ***Tanenbaum and Woodhull, 2006).***

The Inferno operating system written in C programming language was designed to support applications written in the concurrent programming language known as "LIMBO" to run across multiple platforms and network. The syntax of this language though influenced by C and Pascal was specifically written for the Inferno environment. It supports data types common to those languages and several higher-level data types; it is easy to use, safe, and more suited to the Inferno environment.

As earlier alluded to, this system was built on three core principles which are perhaps the most

interesting aspects of this operating system and presents user with numerous advantages, these unique method of distributed principles:

Resources as files: Inferno virtually views all resources as a file system for example a TCP/IP network can be manipulated by accessing *puck$ ls /net/tcp/0* or access to the windowing system is via another file *puck$ ls /dev/draw/2*. The preceding examples demonstrate how all resources are represented as files within a hierarchical file system. In other words, all Inferno resources, both local and remote such as storage devices, processes, services are all represented as a file accessible by any application by manipulating the relevant files required using standard operations **(Distributed Programming with Inferno, Larry Rau – emphasis added).**

Namespaces: This principle deals with the presentation of the application view of the network as a single, coherent namespace that appears as a hierarchical file system but represents physically separated (locally or remotely) resources – in simple words, the namespace is dynamically adjusted. A key advantage with this is the ability of an application being able to use services and/or resources transparently by building a unique private view of the resources and services it needs to access with each set of resources represented as a hierarchy of files and accessible via the standard file access operations **(Building Distributed Applications with Inferno and Limbo, Phillip Stanley-Marbel).**

Standard Communication Protocol: The third principle is a standard protocol used to access all resources, both local and remote. The representation of all resources as files makes it possible to utilized only one protocol to communicate with and provide resources, both local and remote. This protocol is a file service protocol called *9P* (an earlier variant was called *Styx*) which was developed for the Plan 9 Operating System **(Inferno Design Principles,** www.vitanuova.com**, The Styx Architecture for Distributed Systems, Pike and Ritchie for further reading on 9P and Styx).**

Though this OS can be implement in different environments (as a Native OS or VM), the Inferno architecture comprises of three main layers namely Application, Kernel and Hardware

Inferno Operating System

(see figure 1),

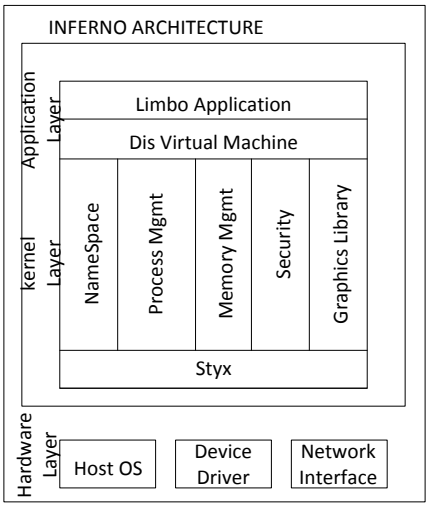

figure1.

(Diagrammatic view of the inferno operating system architecture, Distributed Application Development with Inferno, Ravi Sharma)

Kernel Layer

The kernel layer is responsible for the connection of the application layer to that of the hardware. Memory and Process management occurs at this layer within Inferno OS, with programs in Inferno using the "time-share" feature it possesses; each program has independent access to memory.

Load into memory is the first step in starting an OS. Memory allocation and usage concept within Inferno is somewhat unique; with this operating system not utilizing address translation, virtual and physical memory are the same. Unlike most systems, there are no fixed assignments of components in the memory space, when a request for memory is made; memory is simply sliced up into arenas based on the size(s) of request. In the kernel, blocks of memory are allocated from the main, heap and image pool. Memory allocation in Inferno kernel are made using the *malloc ( ) call* but all calls handled by the function *dopoolalloc ( )* which is define in the *emu /port/alloc.c*. Below is an illustrated example of this function (***code borrowed***)

```
static void *dopoolalloc (Pool *p, ulong asize, ulong pc)
{
```

p here points the *Pool* structure, *asize* gives the size of the allocation requests in bytes and *pc* is

the program counter of the caller **(Principle of Operating System "Memory Management" B. L Stuart, 2009).**

Process management also occurs at this layer in Inferno though unusual in comparison to other traditional OS owing to Inferno's use of a virtual machine, this VM interpreter which is embedded in the OS has more control over user programs than that of a typical OS. Inferno makes use of kernel process internally in addition to user processes though all processes are best thought of as threads which means they all share a common memory space and make up a simple multithreaded program. Kernel process state in hosted Inferno are handled by the host OS in contrast to native Inferno where it is responsible for managing these process, however, the *kproc ( )* function is peculiar to both **(Principle of Operating System "Memory Management" B. L Stuart, 2009).**

Hardware Layer

This layer of the Inferno like other traditional OS provides device driver interface which programs use to connect to the underlying hardware, in simple words, communication between devices and applications resident on Inferno or the operating system itself occurs here. Inferno handles this process by creating or implementing a file tree with a file name, this name is bound to locations in the namespace which provides the files of the console device.

Application Layer

Within the application layer are all Limbo applications, built of modules and are/can be assessed dynamically. Applications built on Limbo unlike those in C language with emphasis mainly on performance and efficiency than security, offers boundary checks which prevent the vulnerability against buffer flow attacks.

Inferno Operating System

Inferno OS offers many advantages as a distributed operating system, as earlier mentioned, its applications are built in the programming language Limbo which supports multitasking and allows the scheduling threads of control. The binary representation of this language assures that applications built can run in different platforms and the Inferno universal environment employed in the architecture of this OS means that identical applications can run under any instance in a distributed fashion. The communication protocol (Styx) employed by Inferno is independent of any underlying network protocol; this ensures that data security is enforced at a single point. The scalability of Inferno is evident in its' ability to present all resources as files, the only functions required by service users include *open ( ), read ( ), write ( ).* Needless to mention that Inferno despite all its advantageous features as a lightweight, scalable and efficient distributed operating system, does possess a few flaws, one of which is poor latency.