Service Dedicated Operating System

Abstract— **In order to limit the number of physical servers, virtualization splits the hardware resources into multiple virtual machines, each dedicated to one service or application.**

A complete operating system should be installed on each virtual machine.

Why multi-tasking and multi-purpose operating systems are used for running one single application?

If the operating system crashes, all services will be unavailable or impacted. Virtual machines limit trouble spreading.

The system that I propose will have to be configured statically and automatically according to the capabilities of the physical system and the profile of the application that will have to be deployed. Only the components necessary to operation of the application will be installed. The code of the system and application will be stored on a partition in read-only mode. This operating system will be transparent for application, providing a traditional system call interface in order to avoid porting the applications.

I. The server issue

A. Goals of virtualization

In order to limit the number of physical servers, virtualization splits the hardware resources of one server into multiple virtual machines, each dedicated to one service or application.

A complete operating system (Windows or Unix) should be installed on each virtual machine. Why these multi-tasking and multi-purpose operating systems are used for running one single application?

Reliability is the main answer. If the operating system crashes or some kind of trouble occurs, all services will be unavailable or impacted. Virtual machines limit trouble spreading.

If a multi-tasking operating system cannot be considered as reliable, why do we install such systems if just a reduced set of features is mandatory in order to run a single application?

B. Server performance

Classic operating systems (Windows, Unix) have been designed for workstations, not for servers (particularly Windows).

Activity and consequently load forecasting is not possible when resource allocation is dynamically performed. Allocating and moreover freeing resources are time consuming, typically dynamic socket creation and destruction for client-server applications.

Dynamic resource management is not more mandatory while running a single application on a dedicated server. At boot time, all available resources will be allocated for the single application.

C. Complexity and safety

The complexity of universal systems like Unix and Windows not only degrades the performances, but also the stability and foresee ability of the system [1]. And that is not improving. The complexity of the systems (operating systems but also information systems) grows more quickly than the performance of the equipments [2].

D. Market history

Server dedicated operating systems already exists. Novell Netware is well-known and performed well when just used for what it has been originally designed: file sharing or printer sharing. But it did not succeed in sharing access to applications maybe because it tried to host them on the same platform.

The other issue is weight of Windows and Linux. Is it possible and relevant today to propose a new OS ? BeOS, PLAN 9, Inferno are good operating systems, but did not really meet the market.

II. Proposals for a lightweight operating system

A. Overview

The system will have to be configured statically and automatically according to the capabilities of the physical system and the profile of the application that will have to be deployed. Only the components necessary to operation of the application will be installed. Once configured, the system will have to require only minimal administration. The code of the system and application will be stored on a partition (or a media) in read-only mode. If possible, this operating system will be transparent for application, providing a system call interface like a traditional system (Unix or Windows) in order to avoid porting the applications for this new OS. This OS will have to be distributed to ensure load balancing and fault-tolerance for the application.

B. Architecture

This architecture includes 3 distinct parts:

1. The profiler
2. The installer
3. The operating system

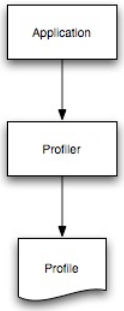
The profiler will be used prior installing a server for a new application. The installer will be used for installing a new server (or a new virtual machine) or a new version of the operating system but for the same application.

1) The profiler

The profiler is a tool which goal is to observe the runtime behavior of a particular service (for example a mail or a web server).

This tool will make an inventory of all system resources required by the target software:

- System calls
- I/O
- Memory allocation sorted by size and number of blocks
- Number of processes
- Number of sockets, …



The result is an XML file called profile that will be used later by the installer. An example of output could be:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_03" class="java.beans.XMLDecoder">
 <object class="Profile">
  <void property="memoryAllocations">
   <void index="0">
    <object class="MemoryAllocation">
     <void property="blockSize">
      <int>1024</int>
     </void>
     <void property="numberOfBlocks">
      <int>24</int>
     </void>
    </object>
   </void>
   <void index="1">
    <object class="MemoryAllocation">
     <void property="blockSize">
      <int>2048</int>
     </void>
     <void property="numberOfBlocks">
      <int>10</int>
     </void>
    </object>
   </void>
  </void>
  <void property="numberOfProcesses">
   <int>50</int>
  </void>
  <void property="numberOfSockets">
   <int>100</int>
  </void>
  <void property="systemCalls">
   <void index="0">
    <object class="SystemCall">
     <void property="callName">
      <string>fopen</string>
     </void>
    </object>
   </void>
   <void index="1">
    <object class="SystemCall">
     <void property="callName">
      <string>fclose</string>
     </void>
    </object>
   </void>
  </void>
 </object>
</java>
```

The profiler should be able to run under Unix and Windows. Of course, this kind of tool already exists, and the profiler is more a matter of software integration than development.

System calls are tracked in order to detect and store the relationships between the application and system components. Later on, the installer will reuse it in order to grant permissions for the application, to call these components.

2) The installer

The installer is managing 2 distinct tasks: building the system and installing it.

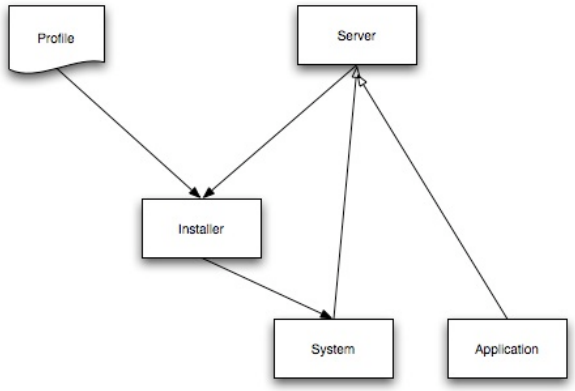
Building the system is the most complex step; it will be designed like an OS factory using a library of OS components or patterns.

Once booted on the target server equipment, the installer lists all hardware resources:

- Number and type of processors,
- Memory size
- Disk capacity
- Network interfaces
- …

It then recovers the profile corresponding to the application that the system administrator wishes to install on this server. According to the profile of the application and hardware resources available, the installer creates the system and installs it on the server as well as the application.

An upgrade of the operating system does not mandatory requires to perform this step again. But installing a new version of the target application is like installing it once since the profile could be different.



3) The system

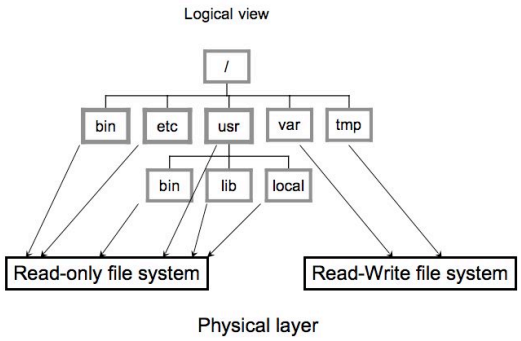
Once the system installed, it starts while launching all the programs and services required in order to run the server and the application. No more programs will then be launched. Further disk access relate only to application data. Thus if something goes wrong because of bad dimensioning or incompatibilities between components, we will be warned generally at boot time and not during normal operation conditions.

I do not plan to use an hardware abstract layer inside the kernel. The first implementation will be for an Intel platform, and since I hope that the code for the system should not be too large, porting onto another platform should not be too long.

a) The File System

The file system is divided into 2 layers:

- A logical layer that presents a classic view of the target file system for which our system has been configured (NTFS or a Unix file system),
- A physical layer that maps each entry of the logical file system either on a read-only drive for the programs and configuration files or on a read-write drive for the application data.



The main goal for such a file system is security without too much complexity. Simple rules like:

- A program cannot be modified or removed during normal operation of the server,
- A configuration file neither,
- Only logs and temporary files are allowed to be created or extended, but only by the registered service granted by the profile

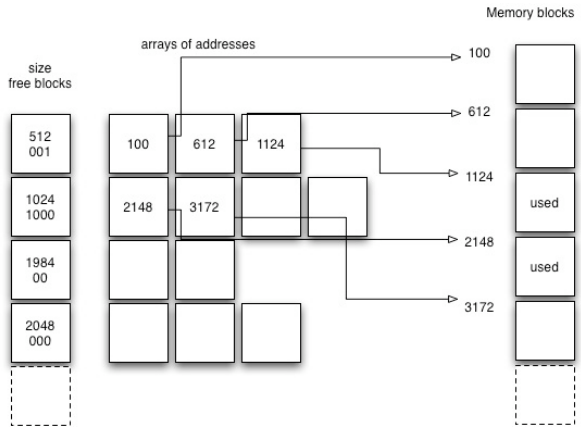
One collateral benefit, after installing and configuring the system, is to engrave it onto a bootable CD. In fact, on the demonstration platform, we choose the ISO CDFS file system in order to implement the read-only part of the physical layer. In case of crash (power failure) , it will be easy to automatically repair the system.

b) Memory Management

The profiler spied the memory allocations performed by the target application. Since we know exactly how many blocks of various sizes are required, the memory manager should be designed as simple as possible for speed efficiency. Typically, the data structure chosen in order to store the list of free blocks will be static.

The final implementation is not yet decided but could be like this:

- An hash table indexed by the block size will store arrays of free blocks for this block size,
- A bit map will indicate which blocks are free,
- Each array will store the pointers to the block in the memory heap
- The size of each array is determined according to the profile for this application



All blocks will be allocated before launching the application.

Since we know how many different block sizes we will manage, the number of keys in the hash table is known and thus the size of the hash table. This information will be used in order to choose the optimal hash code function.

While running the application, 2 events may occur:

1. Unexpected block size,
2. No free block for an expected block size.

For both cases, a classic dynamic memory management algorithm will relay the static memory manager. Statistics of dynamics allocations versus static allocations should be logged in order to detect an inaccurate profile.

c) The API

Like the file system, the API will be application dependant. If the target application was supposed to run under Linux, our system will present a system call library compliant with Linux. A library function will map the system call with the corresponding kernel feature.

Thus, our system does not belong to the exo-kernel family; our goal is to mimic a classic operating system without modifying neither the source code nor the binary code of the application.

But like for an exo-kernel architecture, I will have just to rewrite the functions relevant for most applications.

d) The scheduler

Once again, we will try to get a benefit from the knowledge provided by the profiler.

First, the profile will list all the required features that will be present in the kernel or as daemons. Thus the installer will provide the exact dimension for all the queues used for task management.

Second, the profile reveals which kernel features or daemons are called by the application and how many times. For safety and security, while under normal operation, the kernel will not allow a system call that has not been forecast according to the profile.

I do not plan to design a real-time operating system. I am looking for speed performance, not for predictive scheduling. But some tiny real-time systems could be really efficient while running on a classic PC architecture instead of device with limited resources.

III. Security and safety issues

The main safety and security feature is provided by the read-only file system along with the matrix of authorized system calls. Two other features will contribute to global security:

First, there is no user (and group) management except for applications that requires it (for examples Ingres or Globus Toolkit). The only local real account is for the administrator in order to shutdown the system.

Second, the installer will only install the commands et the system calls required by the application. Typically, neither the command ‘chmod” nor the ‘chmod system call will be available during normal operation of the server in order to avoid compromising the system through a possible security hole in the application.

IV. Conclusion

I am writing the technical specifications (including the file system and the memory management). A demonstration platform for the file system management has already been developed in Java. This platform has been developed in Java language. Under Windows XP, we were able to list a directory which files were located on different devices: the hard disk, a USB key and a CDROM.

The first goal is to achieve a real platform in order to benchmark the system for a typical service (Apache web server or exim mail server).

References

[1] Gary McGraw, *Software Security : Building Security In*, Addison-Wesley, January 23, 2006

[2] Jacques Printz, *Puissance et limites des systèmes informatisés*, Hermès/Lavoisier, 1999