Principles of Computer Science II

Virtualization

Ioannis Chatzigiannakis

Sapienza University of Rome

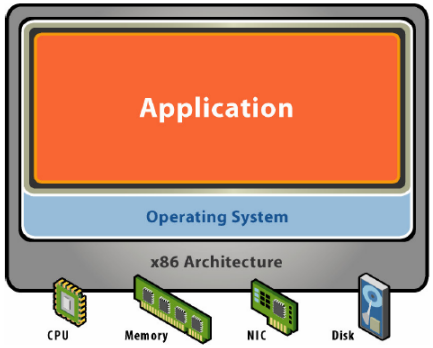
Lecture 9

Virtualization

- Virtualization deals with "extending or replacing an existing interface so as to mimic the behavior of another system"
- Virtual system examples:
  - virtual private network,
  - virtual memory,
  - virtual machine,
  - ...

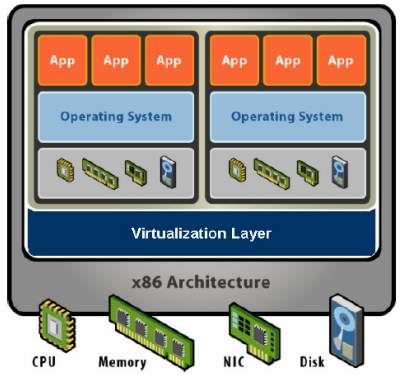
Starting Point: Physical System

- Physical Hardware
  - Processors, Memory, I/O devices, ...
  - Physical resources often underutilized
  - Periods that are over-utilized
- Software:
  - Tightly coupled to Hardware,
  - Single active OS,
  - OS controls Hardware



What is a Virtual Machine?

- Hardware-level Abstraction
  - Virtual Hardware: Processors, Memory, I/O devices, ...
  - Encapsulates all OS and application state.
- Virtualization Software:
  - Extra level of indirection decouples hardware and OS,
  - Multiplexes physical hardware across multiple "guest" VMs,
  - Strong isolation between VMs,
  - Manages physical resources, improves utilization.

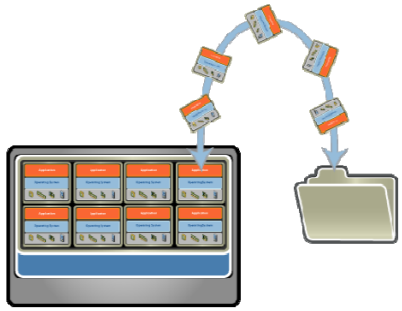


Virtual Machine Isolation

- Secure Multiplexing:
  - Run multiple VMs on single physical host,
  - Processor hardware isolates VMs.
- Strong Guarantees:
  - Software bugs, crashes, viruses within one VM cannot affect other VMs
- Performance Isolation:
  - Partition system resources,
  - Example: VirtualBox controls for reservation, limit, shares.

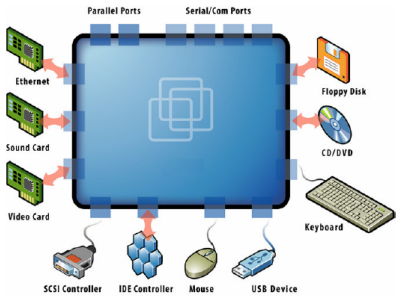
Virtual Machine Encapsulation

- Entire VM in a file:
  - OS, applications, data;
  - Memory and device state.
- Snapshots and Clones:
  - Capture VM state on the fly and restore to point-in-time,
  - Rapid system provisioning, backup, remote mirroring.
- Easy Content Distribution:
  - Pre-configured apps, demos.
  - Virtual Appliances.



Virtual Machine Compatibility

- Hardware Independent:
  - Physical hardware hidden by virtualization layer,
  - Standard virtual hardware exposed to VM.
- Create Once, Run Anywhere:
  - No configuration issues,
  - Migrate VMs between hosts.
- Legacy Virtual Machines:
  - Run legacy OS on new platform.



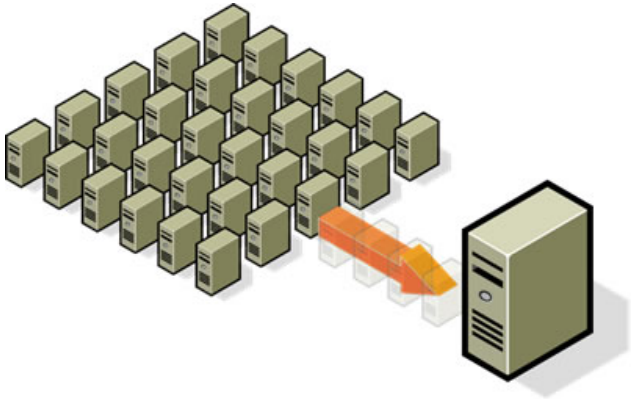
Common Uses

- Test and Development
  - Rapidly provision test and development servers.
  - Store libraries of pre-configured test machines.
- Business Contunuity
  - Reduce cost and complexity by encapsulating entire systems into single files
  - Replicated and restored on demand into any target system.
- Enterprise Desktop
  - Secure unmanaged PCs without compromising end-user autonomy by layering a security policy in software around desktop virtual machines.

Common Uses

- Run legacy software on non-legacy hardware
- Run multiple operating systems on the same hardware
- Create a manageable upgrade path
- Manage outages (expected and unexpected) dynamically

Virtualized Data Centers



Reduce costs by consolidating services onto the fewest number of physical machines

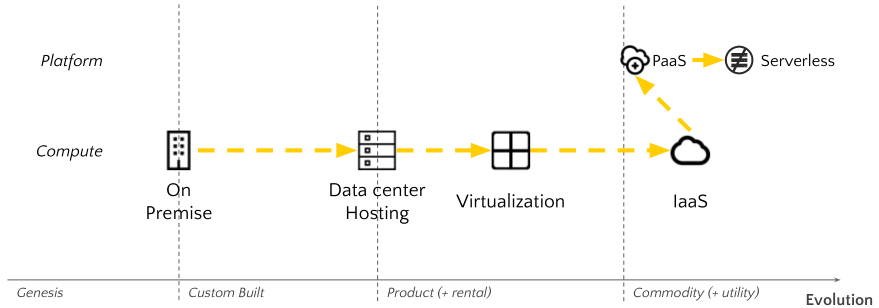
Non-virtualized Data Centers

- Too many servers for too little work
- High costs and infrastructure needs
  - Maintenance
  - Networking
  - Floor space
  - Cooling
  - Power
  - Disaster Recovery

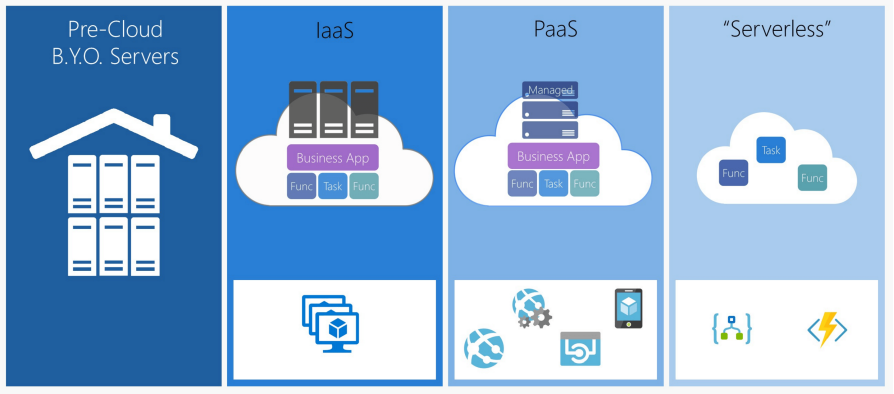
Dynamic Data Centers

- Virtualization helps us break the "one service per server" model
- Consolidate many services into a fewer number of machines when workload is low, reducing costs
- Conversely, as demand for a particular service increases, we can shift more virtual machines to run that service
- We can build a data center with fewer total resources, since resources are used as needed instead of being dedicated to single services

Towards Serverless Computing



Function as a Service



Kernel Subsystems

- File system
  - Deals with all input and output
  - Includes files and terminals
  - Integration of storage devices
- Process management
  - Deals with programs and program interaction
  - How processes share CPU, memory and signals
  - Scheduling
  - Interprocess Communication
  - Memory management
- UNIX variants have different implementations of different subsystems.

What is a Shell?

- The user interface to the operating system
- Functionality:
  - Execute other programs
  - Manage files
  - Manage processes
- A program like any other
- Executed when you "open a Terminal"

Shell Interactive Use

- The # is called the "prompt"
- In the prompt we type the name of the command and press "Enter"
- The prompt allows
  - Command history
  - Command line editing
  - File expansion (tab completion)
  - Command expansion
  - Key bindings
  - Spelling correction
  - Job control

Prompt: The Command Line

```
# date
Sat Apr 21 16:47:30 GMT 2007
```

Error Handling

- If we type a wrong command, an error message appears

Prompt: The Command Line

```
# datee
datee: no such file or directory
```

- The error message states that either the file or the folder (directory) was not found
  - In the prompt all commands are assumed to be connected to a file ...
- The arrow keys ↑ ↓ allow to look-up previous commands
- The arrow keys ← → allow to move within the same command line

Terminating Command Execution

- We can interrupt the execution of a command by pressing *ctrl-c*
- We can "freeze" the output of the execution of a command by pressing *ctrl-s*
  - To "un-freeze" the output of a command we use *ctrl-q*
  - Note – only the output is frozen not the actual execution
- To close a terminal we use *ctrl-d*
  - We may need to press multiple times *ctrl-q*
  - All programs currently running will terminate

Manual Pages

- The command *man* allows to access the manual pages
- Manual pages are organized in categories
  1. Commands – *ls*, *cp*, *grep*
  2. System Calls – *fork*, *exit*
  3. Libraries
  4. I/O Files
  5. File Encoding Types
  6. Games
  7. Miscellaneous
  8. Administrator's Commands
  9. Documents
- We can request a page from a specific category
  man [category] [topic]

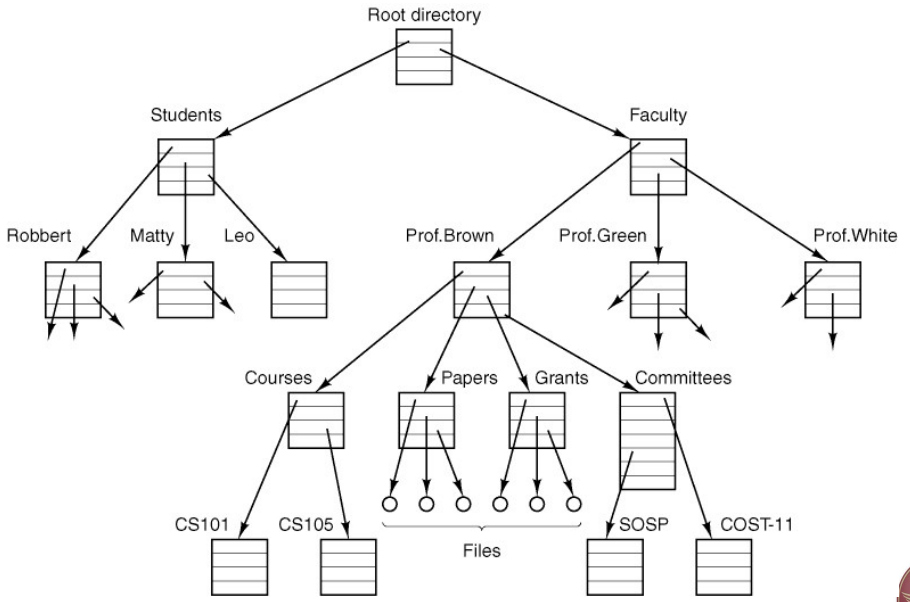
Manual Pages

```
FORK(2)                  Minix Programmer's Manual                  FORK(2)

NAME
     fork - create a new process

SYNOPSIS
     #include <sys/types.h>
     #include <unistd.h>

     pid_t fork(void)

DESCRIPTION
     Fork causes creation of a new process.  The new process  (child  process)
     is an exact copy of the calling process except for the following:

          The child process has a unique process ID.

          The child process has a  different  parent  process  ID  (i.e.,  the
          process ID of the parent process).

          The child process has its own  copy  of  the  parent's  descriptors.
standard-input, 1-24 (Top)
```

File System

- All system entities are abstracted as files
  - Folders and files
  - Commands and applications
  - I/O devices
  - Memory
  - Process communication
- The file system is hierarchical
  - Folders and files construct a tree structure
  - The root of the tree is represented using the /
- The actual structure of the tree depends on the distribution of Linux
  - Certain folders and files are standard across all Linux distributions

File System Example



Standard Folders

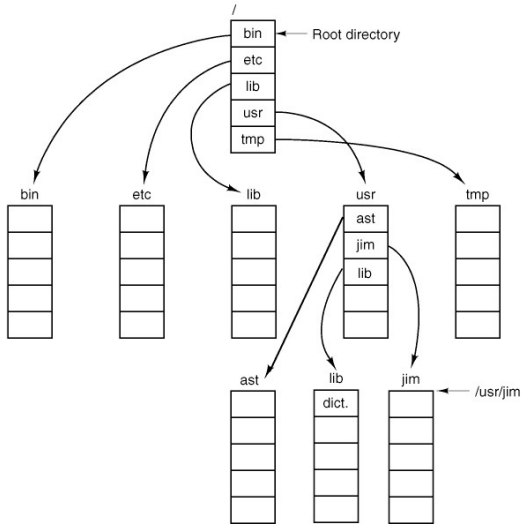
- `/bin` – Basic commands
- `/etc` – System settings
- `/usr` – Applications and Libraries
- `/usr/bin` – Application commands
- `/usr/local` – Applications installed by the local users
- `/sbin` – Administrator commands
- `/var` – Various system files
- `/tmp` – Temporary files
- `/dev` – Devices
- `/boot` – Files needed to start the system
- `/root` – Administrator's folder

Example of File Metadata

```
# ls -la
lrwxrwxrwx  1 bin    operator   2880 Jun  1  1993 bin
-r--r--r--  1 root   operator    448 Jun  1  1993 boot
drwxr-sr-x  2 root   operator  11264 May 11 17:00 dev
drwxr-sr-x 10 root   operator   2560 Jul  8 02:06 etc
drwxrwxrwx  1 bin    bin           7 Jun  1  1993 home
lrwxrwxrwx  1 root   operator      7 Jun  1  1993 lib
drwxr-sr-x  2 root   operator    512 Jul 23  1992 mnt
drwx------  2 root   operator    512 Sep 26  1993 root
drwxr-sr-x  2 bin    operator    512 Jun  1  1993 sbin
drwxrwxrwx  6 root   operator    732 Jul  8 19:23 tmp
drwxr-xr-x 27 bin    bin        1024 Jun 14  1993 usr
drwxr-sr-x 10 root   operator    512 Jul 23  1992 var
```

Navigating the File System

- Each folder contains two "virtual" folders
  `ls -la`
  `. ..`
- The single dot represents the same folder
  `./myfile` ⇒ `myfile`
- The two dots represent the "parent" folder in the tree



File System Security

- For each file we have 16 bit to define authorization
  - 12 bit are used by the operator
  - They are split in 4 groups of 3 bit – 1 octal – each
- The first 4 bit cannot be changed
  - They characterize the type of the file (simple file, folder, symbolic link)
  - When we list the contents of a folder the first letter is used to signify:
    - – simple files
    d – folders
    l – symbolic links
- The next 3 bit are known as the s-bits and t-bit
- The last three groups are used to define the access writes for read 'r', write 'w' and execute 'x'
  - For the file owner, users of the same group, and all other users.

File System Permissions Examples

```
Type Owner Group Anyone
  d   rwx   r-x   ---
```

- Folder
- The owner has full access
- All users that belong to the group defined by the file can read and execute the file – but not modify the contents
- All other users cannot access the file or execute it
- To access a folder we use the command *cd* given that we have permission to execute 'x'

Changing the File Permissions

Examples of File Permissions

Binary	Octal	Text
001	1	x
010	2	w
100	4	r
110	6	rw-
101	5	r-x
-	644	rw-r--r--

- The command *chmod* allows to modify the permissions
- There are 2 way to define the new permissions
  1. Defining the 3 Octal – e.g., *644*
  2. By using text – e.g., *a+r*

Some Examples of *chmod*

```
make read/write-able for everyone
# chmod a+w myfile

add the 'execute' flag for directory
# chmod u+x mydir/

open all files for everyone
# chmod 755 *

make file readonly for group
# chmod g-w myfile

descend recursively into directory opening all files
# chmod -R a+r mydir/
```

Changing the Owner and Group of a File

- The command *chown* allows to change the owner of a file
- The command *chgrp* allows to change the group of a file

```
give ownership to ichatz
# chown ichatz myfile

set group to students
# chgrp students mydir/

give ownership to pcs and group to students
# chgrp pcs:students myfile mydir/

descend recursively into directory opening all files
# chown -R ichatz mydir/
```

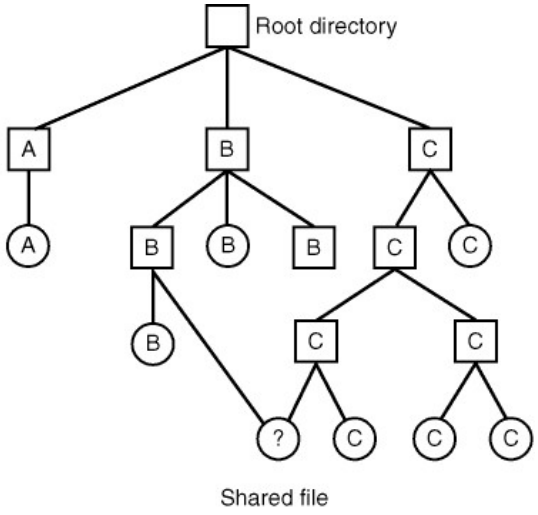
Symbolic Links

- The file system enables to create symbolic links
- Two types are provided
  - Symbolic link
  - Hard link
- The contents and metadata of the original file are used for all operations

```
create a symbolic link to a directory
# ln -s /var/log ./log
# ls -lg
lrwxrwxrwx 1 operator    8 Apr 25  log -> /var/log
```

- The contents and metadata of the original file are used for all operations
  - Except for deletion.

Examples of Symbolic Links



Access Dates

- For each file the system keeps track of
  - Date of last usage/access
  - Date of last change

```
check last usage time
# ls -lu
drwxrwxrwx  1 bin    bin            7 Apr 25  1993 home
lrwxrwxrwx  1 root   operator       7 Apr 25  1993 lib
drwx------  2 root   operator     512 Mar 30  1993 root

check last change time
# ls -lc
drwxrwxrwx  1 bin    bin            7 Apr 25  1993 home
lrwxrwxrwx  1 root   operator       7 Oct 27  1993 lib
drwx------  2 root   operator     512 Oct 27  1993 root
```