Nuclear-fusion reactor smashes energy record

The experimental Joint European Torus has doubled the record for the amount of energy made from fusing atoms — the process that powers the Sun.

Elizabeth Gibney
6.033 in the news

https://www.psfc.mit.edu/research/topics/alcator-c-mod-tokamak
A year long effort concluded with the replacement of three proprietary computers by one computer (DAQS) which is responsible for the bulk of digitizer control, data collection and data formatting [6–7]. This one new computer is running a System V version of UNIX, and has a VME bus and controller connected to existing CAMAC equipment. One enhanced serial highway is installed and a second will be added in late FY95. There were many benefits to be reaped by converting to a fast, reliable UNIX system. These are summarized in Table II.

Table II
Benefits of Converting to UNIX DAQS

<table>
<thead>
<tr>
<th>Benefit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast code development due to increased CPU speed</td>
</tr>
<tr>
<td>Ability to develop on desktop and move to DAQS CPU</td>
</tr>
<tr>
<td>Ability to write window based applications using various available software</td>
</tr>
<tr>
<td>Use of standard languages — largely C</td>
</tr>
<tr>
<td>Cross over skills from other UNIX systems</td>
</tr>
<tr>
<td>Easy development of client/server software, esp. with other UNIX systems</td>
</tr>
<tr>
<td>Ability to use software developed on other systems (specifically the IPCS system for asynchronous messaging developed at LLNL [8])</td>
</tr>
<tr>
<td>Ease of networking and relative speed of TCP/IP</td>
</tr>
<tr>
<td>Compatibility with existing CAMAC equipment</td>
</tr>
<tr>
<td>Ability to configure multiple CAMAC highways (2 highways are 1.99 times as fast as 1 highway as the I/O overlap is nearly total)</td>
</tr>
<tr>
<td>Speedup by a factor of 4 or more in data acquisition</td>
</tr>
<tr>
<td>Availability of database software at a reasonable cost</td>
</tr>
<tr>
<td>CPU cycles available for various compute purposes</td>
</tr>
<tr>
<td>Excess I/O cycles available</td>
</tr>
<tr>
<td>Ease of expanding hardware for our needs (new disks, additional CPUs, more memory, additional ethernet)</td>
</tr>
<tr>
<td>Reliable operation of hardware</td>
</tr>
<tr>
<td>Ease of system maintenance (both hardware and software)</td>
</tr>
</tbody>
</table>
6.033 in the news

Computer systems support all sorts of things, and the design decisions we make affect what those things are capable of.

A big data problem

Each of the team’s ITER simulations consisted of 2 trillion particles and more than 1,000 time steps, requiring most of the Summit machine and one full day or longer to complete. The data generated by one simulation, Chang said, could total a whopping 200 petabytes, eating up nearly all of Summit’s file system storage.

“Summit’s file system only holds 250 petabytes’ worth of data for all the users,” Chang said. “There is no way to get all this data out to the file system, and we usually have to write out some parts of the physics data every 10 or more time steps.”

This has proven challenging for the team, who often found new science in the data that was not saved in the first simulation.

“I would often tell Dr. Ku, ‘I wish to see this data because it looks like we could find something interesting there,’ only to discover that he could not save it,” Chang said. “We need reliable, large-compression-ratio data reduction technologies, so that’s something we are working on and are hopeful to be able to take advantage of in the future.”
Lecture #6: Virtual Machines

even more virtualization, plus kernel designs
operating systems enforce modularity on a single machine using virtualization.

In order to enforce modularity + have an effective operating system, a few things need to happen:

1. Programs shouldn’t be able to refer to (and corrupt) each others’ memory.

2. Programs should be able to communicate with each other.

3. Programs should be able to share a CPU without one program halting the progress of the others.

Today’s goal: run multiple operating systems at once.
**virtual machine monitor** virtualizes the physical hardware for the guest OSes.

- **Virtual machine running guest OS**
- **Virtual machine running guest OS**
- **Virtual machine monitor (VMM)**
- **Physical hardware**
  - U/K, PTR, page table, ...

Guest OSes run in user mode.

Privileged instructions in guest OS will cause an exception, which the VMM will intercept ("trap") and emulate.

If the VMM can’t emulate an instruction, it will send the exception back to the guest OS for handling.

**First question:** What does it mean to emulate?
**virtual machine monitor** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
<tr>
<td>virtual machine monitor (VMM)</td>
<td>physical hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td></td>
</tr>
</tbody>
</table>

**first question:** what does it mean to emulate?

in this example, it means that the VMM needs to step in and translate guest physical addresses to host physical addresses.
virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table

3. physical hardware uses the host page table
**virtual machine monitor** virtualizes the physical hardware for the guest OSes

**first example: virtualizing memory (again)**

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

---

guest OS page tables are marked as read-only memory so that modifications to these page tables also trigger exceptions (and thus allow the VMM can update the other tables)
**virtual machine monitor (VMM)** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

In modern hardware, the physical hardware is aware of both page tables, and performs the translation from guest virtual to host physical itself.

---

Katrina LaCurts | lacurts@mit.edu | 6.033 2022
virtual machine monitor virtualizes the physical hardware for the guest OSes

guest OS

<table>
<thead>
<tr>
<th>virtual hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual machine monitor (VMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

guest OSes run in user mode

privileged instructions in guest OS will cause an exception, which the VMM will intercept ("trap") and 
emulate

if the VMM can’t emulate an instruction, it will send the exception back to the guest OS for handling

figuring out how to emulate an instruction is not enough; we also need to make sure that the VMM is trapping all relevant instructions
virtual machine monitor virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K</td>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
</tr>
</tbody>
</table>

virtual machine monitor (VMM)

physical hardware

U/K, PTR, page table, ...

para-virtualization: modify guest OS slightly

binary translation: VMM replaces problematic instructions with ones that it can trap and emulate

hardware support: architecture provides a special operating mode for VMMs in addition to user mode, kernel mode

figuring out how to emulate an instruction is not enough; we also need to make sure that the VMM is trapping all relevant instructions
**monolithic kernel:** no enforced modularity within the kernel itself

- Application
  - Basic interprocess communication, virtual memory, scheduling, file server, device drivers, network, ...

- Hardware

**microkernels:** enforce modularity by putting subsystems in user programs

- Application
  - application
  - IPC
  - device driver
  - network

- Basic interprocess communication, virtual memory, scheduling

- Hardware

Despite the modularity, it’s not clear that redesigning an operating system from a monolithic kernel to a microkernel is a good idea, in part for reasons of **performance**.
**virtual machines** allow us to run multiple **isolated** OSes on a single physical machine, similar to how we used an OS to run multiple programs on a single CPU.

**monolithic kernels** provide no enforced modularity within the kernel. **microkernels** do, but redesigning monolithic kernels as microkernels is challenging.

we have cared about **performance** in all aspects of our operating systems journey so far, and next time we’ll start to think about performance more generally.

you have now seen **virtualization** applied as a solution to many different problems. the details change depending on what problem we’re solving, but the goal of virtualization remains the same.