

Lecture Notes for 02/02/06: Access Control

(Blue part is taken from: http://en.wikipedia.org/wiki/Virtual_Machine)

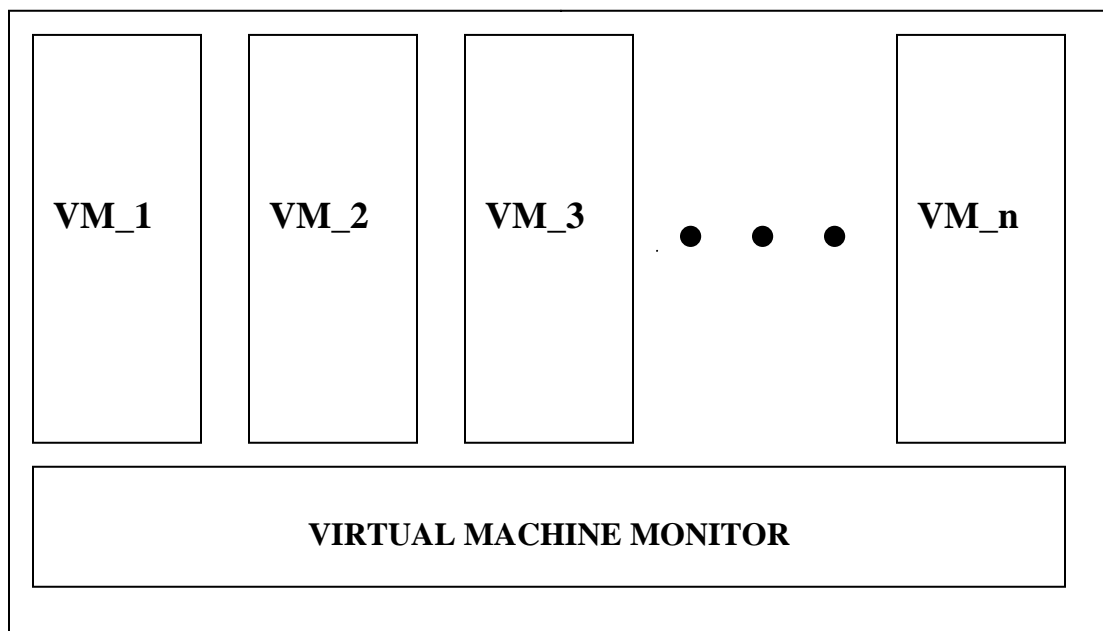
Hardware:

In last class we talked about a computer system that supports a supervisor and that supervisor can control the following:

- Memory accessible to a non-supervisor program. (This is controlled with the help of TLBs, IHA, and supervisor bit.)
- I/O performed by non-supervisors. (We get this one for free because of memory access control.)
- Time on CPU for non-supervisors. (This is controlled because we can have interrupts every 100 or 1000 seconds which would give control back to the supervisor.)

Question) Now, what high level architecture can we build for this computer system?

Answer) We can have the following:



(VM stands for Virtual Machine. The original meaning of **virtual machine** is the creation of a number of different identical execution environments on a single computer, each of which exactly emulates the host computer. This provides each user with the illusion of having an entire computer, but one that is their "private" machine, isolated from other users, all on a single physical machine. The host software which provides this capability is often referred to as a **virtual machine monitor**.)

General information about this model:

- Virtual Memory Monitor (VMM) gives each virtual machine the illusion of a complete computer to itself.
- Each Virtual Machine has its own memory space.
- There is no overlap amongst memory.
- CPU runs each VM for 100th of a second and then switches to the next VM for execution.
- For jobs that require printing from a printer, the CPU has to run each job until it is finished before switching to the next job that is available for printing.
- There exists one big disk, but the disk space is partitioned amongst the VM's.

Sharing In This System:

What can we say about sharing in this system?

- Code: There is NO sharing of code amongst VMs.
- Data: There is NO sharing of data amongst VMs.
- Hardware: All VM's are on a single physical machine.

So, in terms of sharing this is still not so great.

Security in the System:

Which of the security goals are accomplished?

- Confidentiality: This is maintained.
- Integrity: This is also maintained.
- Availability: This is maintained except for when long jobs are using the printer.

So, in terms of security the system is good.

However, there is one security concern. CPU's tend to get hot during long computations and it is possible to measure the heat. By measuring the heat, one virtual machine, say VM_1, might be able to tell that another virtual machine say, VM_2, is doing a lot of computations. Moreover, by using several VM's, the user can get an idea of the scheduling policy, and can figure out which VM has a high scheduling priority.

Trusted Computing Base (TCB) In The System:

What resources do we trust in this system?

- VMM
- Hardware.

Good Design Principles in the System:

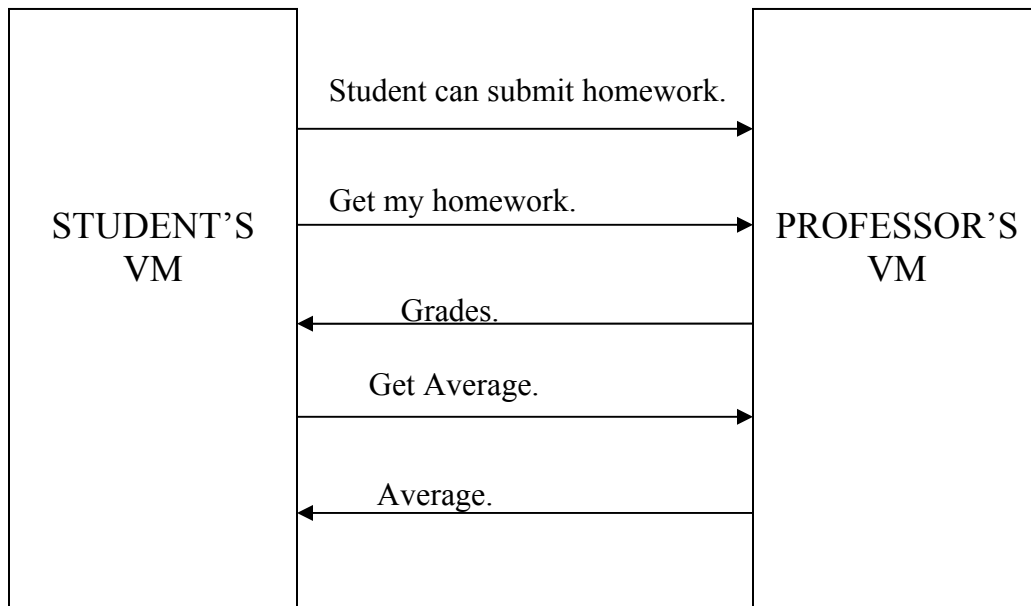
Does the system have

- Small TCB: Yes, the VMM.
- Economy of Mechanism: Yes, The code for this system is simple and easy to understand.
- Complete Mediation: Yes, VMM needs to protect itself and all other Virtual Machines. System depends upon complete mediation.

Changing System for More Sharing:

So far we had VMs that were not cooperating with each other. Let us add a property such that the VMs have more sharing amongst themselves.

Let us take as an example the case in which students would have communication with their professors with regards to their grades:



In this system you can now send messages from one VM to another.

This system has the following aspects;

- Built on Virtual Network.
- VMM ensures the authenticity of the sender ID in each message.

- Client/Server model.

What amount of trust does the two VM have?

- Professor is not confused and is honest.
- Student does not send messages in a huge, irritating amount.
- Trust is just application level.

Remote Procedure Calls:

- Caller and callee can be mutually distrusting. So, we need to build something such that both caller and callee could get what they want even if they don't trust each other.
 - To do this each VM must validate incoming messages.
- This system works:
 - There can be a shared database.
 - There can be shared code for computing the average, etc.
 - There is privacy.
 - There is Confidentiality.
 - There is Integrity because students can't modify their own or other's grades.
 - Availability is also fine.

This system does come at a cost: Each VM has to protect itself from sending or receiving malformed messages.

Who is allowed to do what in this system?

- Professor's VM is enforcing some access control policy.
 - Maybe Ad Hoc.
- Let us formalize access control.
 - We have Objects that professors or students want to access.
 - Subjects \leftrightarrow Domains.
 - We can represent policy as a matrix such that:
 $A[d, O] = \{r \mid d \text{ can perform } r \text{ on } O.\}$, where "r" is an access right, "d" is a "domain" and "O" is an object.

One example of an Access Control Matrix would be:

	Professor	Student1	Student2	Student3	Hw Queue	Student1 Grades	Student2 grades	avg
Professor	Grant	Throttle	Throttle		Dequeue.	Read. Write.	Read. Write.	Grant. Revoke.
Student1*		grant			Enqueue.	Read.	–	Read
Student2*			grant		Enqueue.	–	Read.	Read
Student3*				grant				Read

* Student1 and Student2 are enrolled but Student3 is not enrolled.

Primitive Operations on Access Control Matrix:

- 1) Adding a right: $A[d,O] = A[d, O] \cup \{r\}$
- 2) Removing a right: $A[d,O] = A[d, O] \setminus \{r\}$
- 3) Add_Object(d,O) : $\{ A[d,O] = \{grant, revoke\} \}$
- 4) Add_Domain(d1, d2)
- 5) Delete_Object(d,O)
- 6) Delete_Domain (d1, d2)

It also seems sensible to have conditional operations such as:

grant_right (d1, d2, O, r), which means that “d1 give right ‘r’ to object ‘O’ of ‘d2’, if d1 has the ‘grant’ access for object ‘O’.”

Or symbolically we can say:

$$\begin{aligned} &\text{If } grant \in A[d1, O] \\ &\rightarrow A[d2, O] = A[d2, O] \cup \{r\}. \end{aligned}$$

Similarly we can also have a conditional revoke right:

revoke_right (d1, d2, O, r), which means that:

$$\begin{aligned} &\text{If } revoke \in A[d1, O] \\ &\rightarrow A[d2, O] = A[d2, O] \setminus \{r\}. \end{aligned}$$

Extending Access Control Matrices:

We can have:

$$A[d,O] = \{(r,p) \mid d \text{ has the a right } r \text{ to } O \text{ if } p\}$$

As an example,

Student3 has a right to the object “average.” We can make this right a “conditioned access right” like follows:

$$A[\text{student3, avg}] = \{(read, \text{end_of_semester's_date} \geq \text{current_date})\}$$

Can we determine if an Access Control Matrix can ever go bad?

Question) Given an initial matrix A along with the set of conditional commands and a safety condition ' $r \notin A[d,O]$ ', does there exist an algorithm to determine if A could ever violate this condition?

Answer) The Harrison-Ruzzo Ulman theorem says No. The proof is given by embedding a Turing Machine in A and invoking Rice's Theorem.