6.033 Lecture 5
Operating Systems Part IV

Americo De Filippo

May 17, 2023

The last time has been discussed the Client/Server model, one of the most important preconcept was that teh Client and Server were on different machines, in this lecture we will put the model on the same machine. We are going to transfer the model on a **vitual computer**, a virtual computer has a **virtual memory** and a **virtual processor**. The goal is to make it look like a client/server model.

1 Virtual Memory VM

Why do we want a virtualized memory? A virtual memory is needed for isolate the memory of each module, let's say that some module A can write on the memory of some module B we have lost the isolation of the modules. Another problem is the debugging issues that we are going to face when the memory is not addressed at each module.

Solutions You want to verify the memory access for each module that do not enables A to write in others module's virtual memory (are valid only in a context of a given module). So for each module we are going to create an **address space 32bit** which is virtual. With this organization the module cannot access to the memory location of some other module cause is not allowed to do so.

1.1 Simplified VM Hardware

Look at the slides!
When an instruction try to access some virtual address, the microprocess is gonna send this to the VM system that is gonna resolve this in some physical address. The way that the mapping is done is via something called **page-map**, which is a table from virtual address to physical address. So every module we have a own page-map, the job of the processor is to access only the page-map owner to some specific module. In this way we are going to have a 'separate memory for each module'

1.2 How does the page-map work?

The simpler way of looking at it is like a pointer that tells you where the vitual memory starts for a module A. The problem with this is that we have to have sequential memory for each module. The idea is to take the 32 bit virtual address and we are going to split in two pieces: page number and an offset. So now we are going to store a page of memory at each entry in the table that maps for a physical address, in this way we can have not sequential memory, but instead sparse memory. The offset is used for orients ourself in the page.

1.3 How this data structure are putted togheter?

We need something that has to manage all the infrastacture, and that is called the Kernel.

2 Kernel

At this moment we needed someone that manage this data structure, that is able to create new page maps and look at the module. He must be able to add new module, expands memory etc... This supervisory module is called the **Kernel**. The microprocessor at this point is composed by the PMAR (pointer to modules running) and we are going to add the user/kernel bit (this will tells us which is executing). The module is going to have special permissions in particular on rules will be that only the Kernel will be able to change the PMAR. The kernel is going to be another module that is going to run onto the virtual computer but he is going to have all the pagemaps of the others modules and is going to have a table from module to page-map. Futhermore the Kernel can create new map, allocate new memory in the vitual addressed of some module.

2.1 How do we communicate with the Kernel?

How for example we request at the Kernel the allocation of some more virtual addresses. We are going to add the notion of a **Supervisor Call** (special instruction that invoces the Kernel), he accept a parameter name of a gate function (well-defined entry point for invoce a piece of code into another module). So when a user program execute an instruction supervisor call the special instruction is going to set the user/kernal bit to kernal, set the PAMR to the kernal page-map, save the PC to be the address of the gate function.

2.2 The trusted intermidiet

The kernal is the one that will sit between the client for make them communicate. Suppose we have two processes A and B that want to communicate with each other, we are going to create a special set of supervisor-call for the messages exchange. In this case the Kernel has to verify the correctness of the messages of the two modules.

2.3 VM for shared libraries between modules

This practice is maked really simple with the VM organization, cause the kernel is going to handle this request via allocationg in both the page-maps the address of the physical space of the library. In this case we are going to have shared memory between the module (only the kernel can do that).