6.033 Lecture 2
Operating Systems Part II

Americo De Filippo

May 16, 2023

References

[1] Saltzer, Jerome H. and M. Frans Kaashoek. Principles of Computer System Design: An Introduction (2009): **Sections: 2.1-2.4**

Contents

1 Three fundamental abstractions

Every abstraction has the job of interfacing and of interconnect the name with its abstraction.

1.1 Memory

The memory sometimes called storage is one of the fundamental part of a computer system. The memory from an abstract level could've been seen as performing two main operations: **Read** and **Write**. This two function that the memory performs, are done by a fundamental technique called *naming*, the naming conventions are of incredible use for a good and simple access to the element of a memory.

1.1.1 The things that the memory has to perceive

We must assure an important property for the memory allocation and handling, this fundamental property is READ/WRITE *coherence*, this is a must-have for all system. Without a coherence the system would be impossible to handle cause the undefined behavior will enormous. For this reason we will make sure that the memory coherence will always be respected (hopefully). **The coherence is based on the principle that the Read has to read what last has been written.**

Which thing we must watch ourselves from? There are a lot of different things that can make our memory unclear.

1. Concurrency: when we want (mainly for performance) to access at a single cell with different actors at the same time.
2. Remote storage: when we have a problem of delay access to the memory, this problem will cause a problematic question: **Who lately access to the memory?**. This question can be really difficult to be answered and could cause not few problems.
3. Cell size incommensurate with value size: When we are writing many more generally, when we are reading from a large amount of memory we usually read from different Cells that together combined construct the element. This way for storage data, can cause some kind of behavior that are undesirable. One of which could be that while we are writing on a long memory we start to read from it, this may cause some kind of error in reading.

1.1.2 Memory latency

The memory latency is the time that takes the memory for write and read a block of data, this time is fundamental, cause is one of the thing that draw the between the different types of memory. One of the most famous *storage*, is the hard disk, it's based on having different layers of plates, it accesses the data via mechanical movement of the plates. This technique is perfect for a long time storage, but slow for short time access. So it will have a large memory latency, instead the RAM access the data randomly the memory cells, so for most case it will be one of best suited way for improve memory latency, but poor for long time storage.

1.1.3 Memory names and addresses

Memory have to adopt some naming convention for associating data with their position in the memory, sometimes this paring in the memory is done via some physical coordinates that are perhaps stored in a 2 dimensions array. This way of associating each data with his coordinates or *address* is called associating memory (and it's consecutive most of the time). When we do not want a constrained memory we implement a non-associative memory by adding a layer that will try to make the association in a performance friendly manner. This association in usually implemented via remembering the previous association if already done.

1.1.4 Exploiting the Memory Abstraction: RAID

RAID stands for: Redundant Array of Independent (or Inexpensive), the RAID architecture is implemented via the splitting of a disk in more disks that are accessed via a RAID controller. This kind of memory abstraction has different implementation and has a lot of benefits, that the implementations carries along with them. One of them is concurrency of writing at the same time at different parts of a disk, just because theoretically partitioned. It is also implemented for sharing data across multiple physical hard disks or RAMs.

1.2 Interpreters

An interpreter is a program that go through the code of another program and execute every instruction sequentially. The interpreters acknowledge the instruction checks if that instruction is in his *repertories* and performs it.

1.2.1 Interpreter layer

The interpreters are usually organized in layers. Every layer sees the instruction provided by the lowers. For example Java has an interpreter that sees as instruction to interpreter the lower layer which is the byte-code. So it translates from our java code to the byte code and then the compiler will translate it via the instruction of the lower layer which is the binary code.

2 Naming in computer systems

The naming in computer systems are a fundamental practice, when we use naming properly, we are able to make in practice modularity in a way that is easier to understand. Usually naming is used for have a better understanding of the system, for making some indirect differences between two objects, and for the sharing of a resource between the system. In this case we have two different types of sharing: by value and by reference.

2.1 The naming model

Having a good and efficient naming procedure is one of the things that a computer system must have. When we are dealing with a lot of different structure is useful to have an efficient name system. We define a naming model by three principal structures: a Name Space (the alphabet), a Name-Mapping Algorithm (who does the pairing), Universe of Values (the correspondence). Another important thing is the context, the context is the thing that make an important distinction between the ways we may translate a name in the universe of value of the right context. We usually in computer science have different structure and different ways for lookup for a correspondence. Later we will define 3 principal lookup methods: Table lookup, recursive lookup and multiple lookup.

2.1.1 Default and explicit context reference

When we are dealing with a large model, we could find some name that must mean something different depending on which context they are in. Let's say we want to know the meaning of the word *piano* in Italian. We have different meaning from the context that we are in, *piano* in the music context means the instrument, instead in a car may mean go slow! So we have different meaning from the context we are in, but how we can identify the context? Via explicit of implicit reference. An explicit reference could be the absolute path, instead an implicit reference may be the working directory.

2.1.2 Path names, Naming Network and Recursive Name Resolution

We can usually find out the name or the context of a thing, by recursive resolution of the name. For example in a path name like. /usr/bin/vim, this is a path having the context that could be seen in the root, that is the base of the induction for the recursion. A naming network is simply the hierarchy that organize a system, with links between the various components an example may be the *UNIX* file system.

3 Organizing computer systems with names and layers

Always look for the Least
Surprise Principle

An operating system is organized in layer and of course in procedure of naming. A good and fascinating example of a complex system is the operating system of a personal computer. The personal computer from a high level abstraction could be seen as divided in three layers: the Hardware, the operating system layer, the application layer(browser, games, etc.). Sometimes for performance we *bypass* the operating system layer accessing directly at the hardware layer, this operating can be done only when performing non-dangerous operation.

3.1 A Hardware Layer: The Bus

The buses are the way that the hardware communicates, the bus has 3 components: the address bus, the data bus, the control bus. When the hardware wants to communicate with different components of each, it usually performs a broadcast discovery. Every part is the hardware has a range of addresses that it accept. When someone sends a data (that goes in the data bus) it also associates the address number (destination of the message). So let's say we press a key on the keyboard, when pressing the key, to interpret of the I/O component, sends a message on the bus, making as destination addresses of the message the processor that has the compute something with that information. The control bus is usually for telling at the system to do not use that bus that At that moment is occupied with some byte stream of information, or on the other hand for signalize that the data bus is available.

3.2 A Software Layer: The File Abstraction

The file abstraction is one of the thing that is mostly used in operating system as for UNIX. The file is just an interface for the memory, we manage files via three fundamental procedures: OPEN(), WRITE(), READ(), CLOSE(). This fundamental properties enable the operating system to manage files in a really efficient way. When we open a file we check for the permission of accessing of that file, and we also set the cursor (or called the *file pointer*) to the start of the file. Instead, when we read in the file we take further the pointer, and we copy the data founded. The writing is performed by taking some bytes from the buffer and store them into the file starting at the position of the file pointer (or by the 'write cursor'). The closing procedure just closes the file. The abstraction that we have mentioned before about the UNIX operating system is an interesting one. In the UNIX operating systems each device has a file associated with himself. So that when someone (that has the permission) want to access to some information about that device just go and read from that file, in this way we have a really simple a predictable way of behaving for our system. Perfectly in accordance with the Least Surprise Principle