**2025-05-15-os4-en**

0:00  
Everybody knows what file is, right?

0:03  
Everybody's been using files, at least on their user level, so this should be pretty simple.

0:09  
But from the perspective of the operating system, file is just an abstraction that holds a stream of bytes.

0:15  
So basically a file is a sequence of bytes.

0:18  
That's it.

0:19  
Operating system has no other, no further knowledge of what file is.

0:24  
It just stores chunk of data.

0:27  
And this abstraction is used for other applications to store the data usually in a secondary storage.

0:34  
So usually in a persistent way, which, so the file will survive the power outage, the file will survive the reboot of the computer, etcetera.

0:44  
And also the file can be used to exchange data on more, let's say on longer distances, on more via more elaborate communicating channels, etcetera.

0:57  
Most importantly, file provides a sort of identification and and then again, there is some semantic gap between what you know of files because you identify files via their names and usually via their location within some folders.

1:11  
But the system uses some numeric identifiers.

1:14  
So the operating system for for the operating system, the file is basically a number, OK.

1:20  
And this number is translated actually vice versa name and path which is something you are used to as a users is translated to this number via some via some mechanisms of the file systems.

1:34  
So this way both sides are happy because for humans we still have mnemonic way how to identify the files.

1:43  
Because if you name the file funny video dot MP4 or something, then you know it's a funny video right?

1:50  
Right away.

1:50  
You can identify it, you can remember it.

1:53  
If you identify the file by a huge number, you as a humans wouldn't identify it.

1:59  
And there are, unless there are some cyborgs amongst you.

2:02  
So for the humans, we are happy.

2:04  
And for the operating systems, the operating system is still happy because numbers are much more easier to handle, much more easier to store, much more easier to process.

2:16  
Yeah, some parts of the file name sometimes have special meaning.

2:21  
Again, this is not usually the thing of the core of the operating system.

2:26  
This is not the kernel stuff.

2:28  
But some parts of the operating system like the libraries, like the extensions, like the user environment, which is sometimes integrated or more or less integrated.

2:37  
Part of an operating system can interpret certain file names with certain meaning.

2:43  
For instance, Windows tend to interpret extensions of the files like exa.com dot text or something with certain meaning.

2:55  
For instance, if you have a file with Exa extension, the Windows environment will guess it's probably an application and it can be executed.

3:06  
So it will offer you a way how to run this file as if it was a program.

3:11  
So it suspects it contains the instructions and all the other data of an application, and it tries to load and execute this file as if it was a program if it contains another extension.

3:23  
Some of these extensions are listed somewhere in the registry of the files of the operating system and it uses specific applications.

3:31  
You open TXT file, it will open your favorite text, browser, text editor.

3:36  
If you open JPEG file, it will open some image browser, et cetera.

3:42  
OK, so this was like fundamentals to get everyone on the same ground.

3:46  
Not too much information there.

3:48  
What you should already know, or what you probably seen at these glimpses of, is how the file is handled or how files are handled in programs in your code.

4:00  
So this is AC code using the standard library in C Very low level stuff.

4:05  
How to open and read a file.

4:08  
So basically you need to open the file 1st and that this is the place where the translation between the path between the mnemonic representation that is common to humans is translated to some numeric representation.

4:22  
Actually, not completely, not exactly.

4:25  
In this abstraction we are getting some pointer, so the details are hidden within some C structure and we are getting pointers to this structure.

4:34  
So it's even more elaborate than that.

4:36  
But that's not that important.

4:37  
We are getting some sort of handle.

4:39  
This FP is like a handle.

4:41  
We can use it only to be passed into very specific functions that operate files.

4:47  
OK and I can test it.

4:48  
If I get a new pointer then I know this file wasn't open.

4:52  
Actually it was.

4:53  
There was some error when I was trying to open the file.

4:56  
For instance, the file doesn't exist.

4:58  
That's a common mistake because I got the name wrong or somebody have deleted it since or I don't have permissions to read this file for instance.

5:08  
So something happens, something bad happens and this file wasn't open.

5:12  
So I'm checking whether I get the valid handle or not.

5:16  
OK, by the way these are over here just a crude way how to tell this abstraction that I would like to read the file.

5:27  
There can be W for write if I want to write the file and also there are some other acronyms like a for appending etcetera.

5:34  
So this string somehow represents the mode in which I would like to access the file.

5:41  
And once I pass this, so if I open the file successfully, then I can perform various tasks.

5:46  
For instance, I can see the file.

5:48  
By the way, this abstraction allows me to, let's say, have some secret hidden pointer within this file.

5:56  
If I open it for reading, this pointer points to the beginning of the file because it expects that I tried to read the file from the beginning to the end.

6:05  
If I open it for appending, this pointer will be at the end.

6:09  
So I can start writing at the end of the file.

6:12  
So I can move this pointer, I can I can adjust it as I wish using the seek operation.

6:18  
So by seeking, I will move this internal hidden pointer which is embedded somewhere within this or associated somewhere with this, with this handler to position 42.

6:29  
And this, this mysterious anum, this mysterious constant will tell this function that I'm seeking it from the beginning of the file.

6:40  
So this is like an absolute address, absolute value.

6:42  
Yeah, I can move it relatively or from the end or something not that important.

6:47  
This is just an adjustment of the internal pointer from which this following read operation will start.

6:54  
And this is getting some characters, in this case characters because I've opened it as a text file, but it can also be some buffer of bytes.

7:04  
I can read binary files as well this way.

7:07  
So this is the pointer to a buffer.

7:09  
This is allocated memory.

7:11  
And I'm reading 16 bytes from this file pointer.

7:17  
Easy enough.

7:18  
And finally, when I'm done, I've read what I wanted, I need to close the file.

7:23  
Well, I don't have to, because when my process terminates, the operating system automatically closes all the files this process was holding.

7:31  
So this is not like a mandatory operation.

7:34  
If your application terminates over here, then it's like a maybe slightly redundant, redundant line.

7:41  
But in the end, it's always good practice to be tidy and to clean up your own mess.

7:47  
So it's a good practice to close your own files once you've opened them.

7:51  
So this is like the last one.

7:52  
And after this I won't be able to use this pointer, this this handler anymore.

7:57  
This is analogy which is written using POSIX abstraction.

8:02  
You probably heard about POSIX in the Linux class.

8:04  
That's right.

8:05  
The POSIX is is like a API specification for Unix operating systems.

8:10  
So if you want to use a very low level Unix stuff, then you can use POSIX directly.

8:16  
Basically libc is implemented on the top of POSIX if you're in the Linux files or in Unix environment.

8:24  
Libc is built on top of Windows functions, which are similar, but they have somewhat more awkward names, longer names, but very similar API.

8:33  
So it's implemented over a different API, but in a very similar manner if you're in the Windows environment.

8:39  
So this is like a benefit that the libc is providing to us is a portability.

8:46  
I can write the same code for Windows, for Unix, for technically, possibly for other operating systems as well.

8:53  
So as you can see, these lines are translated almost one to one, in this case 1:00 to 1:00, because when I'm opening the file using the libc abstraction, there is an analogical function open in POSIX and as you can see, it gets similar arguments.

9:11  
Again, it gets the path to the file and also it gets some flags which indicate what I'm going to do with the file.

9:18  
There is only slight discrepancy here.

9:20  
I'm providing it using some awkward string which is defined in libc library.

9:25  
Here there is a set of constants, set of numbers.

9:28  
Actually this is like a macro constant which is provided by the header which I can put here to tell the operating systems.

9:36  
I would like to only read this file, and again, I need to check whether the descriptor here is valid.

9:43  
By the way, the descriptor over here is a number, is an integer.

9:48  
Here is some more mysterious black box structure, and I'm getting a pointer to that, but here it's a normal number.

9:57  
And by the way, you need to know the specification.

9:59  
Yeah, POSIX is somewhat clumsy in this way, because sometimes you need to know.

10:05  
Everybody knows right That if you get a negative number, actually if you get a -, 1, it's an error.

10:11  
Positive numbers are fine, negative numbers are errors.

10:14  
So -1 in this case is.

10:16  
In this case -1 is the only one only number that can be returned.

10:20  
So if -1 is returned as the handler, I know it's invalid handler.

10:25  
I shouldn't use it for operations.

10:27  
I need to handle the error in the same manner like I handled it in over here.

10:33  
And later you can see the the functions are named slightly differently, but basic.

10:38  
Yeah, the order of the arguments is slightly different, but basically it's the same stuff.

10:42  
You just need to look up this in the manuals.

10:44  
So you you proper properly use the proper function with the proper order of the arguments, but it's the same stuff.

10:52  
What's slightly different?

10:54  
Sorry, Yeah, you get it on another file.

10:56  
What's slightly different is that this number is a number, and I can interpret it as such.

11:02  
And this number is actually sequentially assigned.

11:04  
You've already witnessed that when I was showing you the stack trace of a process that redid a file.

11:11  
So when we opened a file, it got this pointer, this handler.

11:18  
The first allocated number was 3 for that, and we established that we already have three open files when we start working, when our application is booted, when our application is started, we already get three open files, standard input, standard output, and the error output, right?

11:38  
Three files are already open for us.

11:41  
And yeah, these file files are somewhat awkward because somewhat different because they correspond to the inputs to determine on the terminal output on the terminal.

11:54  
Yeah, they can be redirected sometimes, but basically the operating system handles them in the same manner it handles files.

12:02  
So remember, standard input is basically a file.

12:04  
It has some special properties.

12:06  
For instance, you can't seek it, you can only read it, You can't, you can't perform seek operations on the on the input, right?

12:12  
So yeah, there are some technical differences, but basically it's still a file descriptor.

12:19  
By the way, what I've shown you on the previous slide was a very simple read operation, open, seek, read and close for operations, if I want to be exact.

12:27  
But basically it was all about reading.

12:30  
Yeah, we have some additional operations like creation of the files.

12:33  
That's obvious because if we cannot create files, there will be no files, right?

12:38  
But the creation is basically merged.

12:40  
Usually it's merged with opening when you're opening a non existent file and you provide specific flags, specific attributes that and the file is created as an empty file.

12:53  
Same goes for deletions.

12:54  
A deletion is somewhat tricky.

12:56  
I will talk about it when I explain how hard links work, because deletion is slightly more complicated and always we can change the size of the file.

13:05  
Normally the file is extended if you write to a file.

13:08  
If if you're putting more data into the file and we're writing after the end of the file, the file is automatically enlarged.

13:16  
So the size is automatically adjusted.

13:18  
But sometimes you want to crop the file, you need to shrink the file.

13:22  
So for that you need to use a specific operation.

13:25  
And also the file has some attributes.

13:27  
I will talk about that in short order.

13:29  
So there are some attributes and we can change sometimes some of these attributes and sometimes actually this goes hand to hand with this point, the files are usually buffered in memory because it would be very tedious.

13:42  
It would be very problematic if every single operation would be translated into the disk access, especially if you're using some slow hard drive or if if the file is stored on a network file system.

13:56  
So in these cases it might be very tedious if every IO operation on the file will translate into the direct access to the storage or the direct access over the network.

14:07  
So there is 2 levels of buffering.

14:09  
At least one level is performed at the system level.

14:13  
SO11 level is at the POSIX level.

14:16  
When you open a file, there is some buffer associated with it and if you're loading or storing data from unto it, the operating system is somehow caching this data.

14:26  
And also, usually your runtime in CC plus plus, C#, Python, whatever language you're using, there is usually another buffer in the runtime which also performs another level of caching.

14:38  
So yeah, some of the data are stored at least temporarily only in the memory.

14:44  
Even if you perform the write operation and you're done with it, then the data are still in the memory.

14:50  
And then it may have not been yet transcribed to the persistent storage.

14:54  
So for that, we have the flush operation, which tells the buffers, hey, right now I would like to synchronize everything that's in the memory with the persistent storage.

15:04  
So I have some reassurance.

15:06  
Samsung guarantees that the data are actually stored on the persistent storage.

15:12  
So when the power goes out or my application crashes, for instance, I will have the data stored properly.

15:22  
OK.

15:22  
By the way, the buffering is a bit tricky because usually we cannot buffer the entire file.

15:27  
Yeah, if you have a very small text file, it's no problem.

15:30  
But if you have a huge file, if you have a huge video, for instance, which can spend gigabytes or 10s of gigabytes, perhaps today if you have 8K quality or perhaps even with some 3D features or whatever, then you can easily have several 10s of gigabytes per movie, no problem.

15:51  
So it would be very difficult to cache that in memory if you have a simple laptop or simple desktop.

15:57  
So for that, it's difficult to tell which part of the file needs to be properly cached because yeah, sometimes you need to read the file from the beginning to the end, like when you are watching the movie.

16:07  
Sometimes you need random access over the file because it's some database or something else which is accessed as needed by some application.

16:17  
By the way, there are some alternatives to the file operations to the API you've seen on the previous slide.

16:24  
Well, in this the the operation operating manual or standard operating procedure if you will, is very simple.

16:32  
You need to open the file, then you're using the handle to perform whatever operations you need.

16:37  
And each of these operation is blocking, which means if you perform this read or this read, it will block, it will pause your execution until the data are ready.

16:49  
So when this function returns back, when control is returned to your main code, to this top level code, you are certain, you may be certain that the data are already in this buffer, everything is there, no problem.

17:05  
But sometimes you need other types of APIs.

17:08  
By the way, there are two, at least the most common types.

17:12  
One is called memory mapping.

17:13  
So you can directly map the entire contents of a file into the memory.

17:19  
I'll get back to this one when I'll explain how memory operations, how virtualization of the memory works, because that goes hand to hand, one with another.

17:29  
And there is also an asynchronous file IO, which is somewhat, let's say not in the mainstream.

17:36  
That will be the perfect definition.

17:37  
It's slightly on the side, but you can use it to perform some operations asynchronously, which means you don't have a guarantees that if you perform a read, the data are immediately there.

17:52  
You just start the read, it starts reading, it starts the transfer and returns to your code immediately.

17:58  
So the transfer is like loading the data on the background and once the operation is completed, you can, you can either pull it, you can, you can make an inquiry whether the operation has completed or you might get some notifications, but that's not that important.

18:16  
I would like just to tell you that there are other APIs.

18:19  
This API is like the most traditional.

18:21  
You can translate this to Python.

18:23  
Actually in Python it won't be 1 to one but but there are similar functions in Python in C#.

18:30  
OK, so those are the beginnings.

18:34  
I talked about file attributes.

18:36  
So let me explain that on an example because examples are always good.

18:40  
So this is a real list of basic statistics and attributes of a file.

18:45  
I've used a stats function from Linux.

18:48  
So I just dumped some information about this file.

18:51  
So let's take a look what's there.

18:53  
So first of all, yeah, size is obvious.

18:55  
So this is size in bytes.

18:56  
So I know this, this file is roughly 11 kilobytes long, slightly over 11 kilobytes.

19:03  
And it's a regular file.

19:05  
Yeah, that's that sounds like a yeah, nothing.

19:09  
But operating system or file systems usually distinguish several types of files.

19:15  
At the very least, they need to distinguish files and directories.

19:18  
Directories are specific types of files.

19:20  
We'll get to that.

19:21  
So this is a regular file, not a directory, not a special system file or special block file or something.

19:29  
And this is this is some identification, this identifies the device, the hardware device on which the file is stored.

19:41  
And this is the numerical identification I talked about at the beginning.

19:45  
So this is the number under which the operating system can load the file, recognize, recognize the file, identify where where the data of the file are stored in the persistent storage, etcetera.

19:57  
So this is the important number for the operating system, OK.

20:03  
This is how many links are there for the file.

20:05  
I'll get to that later when I explain what links are.

20:08  
So the links are also counted as an attribute.

20:11  
And now I'm getting to the more more interesting part.

20:14  
UID and GID is a user ID and group group ID.

20:19  
Those are identifications of the person and group of persons who own the file.

20:25  
So as you can see, you probably haven't noticed that this, this is my login.

20:29  
So I'm the owner of this file and it, it belongs to my group, my personal group.

20:35  
No, no problem there.

20:36  
And these are the access rights.

20:38  
You've probably already covered this in Linux class, right?

20:42  
So you know what the RW and X attributes 3 times repeated mean, Is that so?

20:48  
Yeah, some people are not doing so.

20:50  
I leave that to Linux classes because it's, it's too difficult to explain in all the detail, but basically it tells me whether I can read the file, write the file or execute the file.

21:01  
And finally, there are three times the access time, modify time and change time.

21:07  
So yeah, the access time is obvious when the file was last accessed, when somebody tried to read the file the last time.

21:15  
By the way, this is, this is one of the attributes that may not be updated all the time.

21:20  
Because if you need to update the access time every time somebody reads a file, it's a huge, it's a huge performance hog.

21:29  
It's it's like a very demanding on the, on the underlying structure on the file system and everything because you're updating basically everything, all the meta information are updated all the time because the operating system is accessing so many files all the time.

21:44  
So sometimes you turn this off if you need the performance, you usually turn this access time off.

21:50  
It's usual.

21:52  
It's not that uncommon practice in many systems and modify time and change time are the times where the file was changed for the last time.

22:01  
Everybody understands the difference between modify and change.

22:06  
Really.

22:06  
I don't remember this one because modify and change it sounds like the same, right?

22:13  
At least to me.

22:14  
So yeah, hint, the first one is when the file content, when the data inside the file changed for the last time, and the second one change is when the attribute, the, I know the metadata were changed for the last time.

22:32  
Don't worry if you don't remember that.

22:33  
I don't remember that either.

22:34  
I always need to look that up because it's like unrememberable, but that's how they named it, so we need to stick with it.

22:43  
So the thing you need to remember is that there are two different time stamps, right?

22:48  
One time stamp is for the modification of the data itself, and another time stamp is there for modification of the attributes and other metadata of the file.

22:56  
That's it.

22:58  
OK, so this is the basic examples of most common attributes of a file.

23:03  
Again, the, the, the exact attributes depend on what file system you're on, what operating system you're using, etcetera.

23:10  
But it's fair to say that there are some metadata, some attributes that cover some time stamps regarding the file and also some privileges and ownership of the file that are very common.

23:24  
OK, I promise that I will say a few things about directories.

23:28  
So first of all, directory, as I've already mentioned, is a special type of file, basically the operating system itself, the kernel of the operating system doesn't distinguish between a directory and the file to to the operating to the kernel.

23:43  
It's still a sequence of of bytes somewhere on the hard drive, somewhere in the persistent storage, but the library is slightly above can read this specific file and interpret it.

23:55  
Because directory is a file that holds a collection, a list of entries that's a term entry.

24:03  
And the entry is basically a file name and the pointer to the file on the hard drive.

24:09  
So this is like a list of files which are stored within or listed within this particular directory.

24:17  
But usually the entry holds no data.

24:20  
The entry is only metadata that refers to the file, usually using the numerical representation like the inote over here.

24:29  
So in the Linux you can imagine the entry as a translation table between the file name and the inote in the very simplistic way.

24:39  
OK, why is why is this stored like this?

24:43  
Yes, first of all, it's much more efficient for look for users and sometimes even for operating system to organize files into directories, right?

24:52  
Because if you have like tons of files, it's better if you get them sorted into different directories.

24:57  
One directory is with with your stuff you need for for studying on Modfis, right?

25:03  
You have someone 1 subdirectory for programming class and other subdirectory for for Arduino classes etcetera.

25:10  
One sub directory for fancy videos right?

25:13  
So you need to organize your data somehow.

25:16  
Another thing is this goes hand to hand.

25:19  
You need to name these boxes.

25:20  
So directories can also have names and sequence of directories forms paths.

25:26  
So you can have a hierarchical organization, and this goes hand to hand with the grouping because you can aggregate files that belong somehow together into one location.

25:37  
This is not only important for you as users who need to organize the data, but it's also important for the operating system libraries, operating system tools, applications, because usually one application has all its stuff in one directory, etcetera.

25:55  
Yeah, the entry, the directory entry may or sometimes may not.

26:00  
It depends store some of the file attributes.

26:04  
So the entry may not be only a simple translation between a name and the inode number, but sometimes it can also contain some of the some of the attributes, depends on the design of the file system.

26:17  
For instance, in fact, which I will show in short order, most all of the attributes are within the entry.

26:27  
On Linux file systems like XE, XD, the attributes are somewhere else.

26:32  
The entry is really only a translation between a name and the inode and that's it.

26:36  
So it depends on the design of the of the file system.

26:41  
So yeah, to work this properly, we need some additional operations for for directories, Yeah, again, we need to create and delete.

26:50  
That's kind of obvious because if if we have a directory, we need a way.

26:55  
If we have directories, we need a way how to create them and delete them.

26:58  
That's obvious.

26:59  
Tricky part is the delete.

27:01  
Deletion of the directory usually means that you cannot, you either need to have an empty directory.

27:09  
If the directory isn't empty, you couldn't delete it.

27:12  
So either you have empty directories and the delete operation delete only empty directories or the deletion simply releases the files.

27:24  
But usually you cannot do that because directories may hold sub directories.

27:29  
So the way the the operating system or the operating system tools go around about it is that you can only delete empty directories.

27:38  
This is like a rule.

27:39  
And if you need to delete the directory with all the files within, then you need some special tools but some additional library which actually performs A recursive search of the directory.

27:50  
It deletes all the files and all the sub directories within recursively and then once the directory is empty then it performs delete operation.

27:59  
So if you're just using your operating system and you hit a delete button on your directory on a directory, it doesn't perform single operating OS operation.

28:11  
It's not a single OS call.

28:14  
It's actually very very huge number.

28:17  
If there are many files within, it may be very huge number of system calls because you need to search all the directories, delete all the files, then delete all the sub directories and then finally you're able to delete the directory itself.

28:31  
By the way, renaming on the other hand is very simple.

28:37  
First of all, renaming within the same directory, just changing the name, it's, it's trivial.

28:44  
It's just a changing 1 substring in in one entry.

28:47  
So renaming operation is very fast and very trivial.

28:51  
And furthermore you can perform rename as a move operation, right?

28:56  
Rename and move are somewhat going together.

28:58  
Most operating system tools are sometimes named renamed, sometimes named moved, and they do the same thing, right?

29:06  
If you perform MV operation in Linux, it sometimes perform rename.

29:12  
If you do this within the same directory, if you provide complete paths, it actually moves the file from one directory to another directory.

29:21  
But the benefit is that moving one file on one file system.

29:27  
If you're trying to move a file to a different file system to a different device, for instance, that's another story.

29:34  
But if you're moving a file within the same file system, it's a trivial operation.

29:40  
It's just taking one entry from one directory and putting it into another directory.

29:45  
And even more, POSIX guarantees this to be an atomic operation, so moving files within the same file system is very fast and atomic.

29:56  
That's beneficial.

29:57  
It's good to know because if you're trying to do something which needs to be safe from multi processing or multi threading point of view, this is one of good beacons that you can use to synchronize your work.

30:11  
And of course, yeah, you can list the members of the directory.

30:16  
That's like obvious feature that needs to be there.

30:19  
Goes without saying.

30:21  
An example, this is how a directory entry looks on the set FAT file system.

30:27  
By the way, does anybody know what FAT file has anybody used FAT file system?

30:33  
At least few Yeah, you probably using it without realizing if you are still using USB sticks.

30:38  
I'm not sure if anybody is still using USB sticks.

30:42  
Perhaps you have a perhaps you have a memory card in your cell phone or in your camera.

30:47  
If anybody uses still cameras, then yeah, probably the the the the SD card in your cell phone or in your camera may still use or is probably still using FAT.

31:00  
Yeah, this is just the general FAT.

31:02  
We are using modern VFAT extension of FAT 32, but that's our just technical details.

31:08  
You don't need to bother with them.

31:11  
So this is like one entry looks like, and actually this is like one entry which is composed of three entries.

31:17  
I will explain in short order.

31:19  
In the old FAT system, there was a limitation that a file name has 8 characters plus 3 characters for extension.

31:29  
That's it.

31:30  
So you are limited by the size.

31:34  
Your file name may have only 8 characters.

31:38  
That's it.

31:38  
Nobody will ever need more than 8 characters, right?

31:41  
So remember that when you're naming your next document.

31:44  
This is the document that I've created On this date and it contains this, so remember that.

31:51  
And FET needs to somehow remain backwards compatible with MS-DOS who was still relying on this and also needs to support newer operating systems like Windows.

32:05  
So they need to make it compatible both ways.

32:07  
And what they did is that if you need a larger, a longer name for a file, it spans over multiple records.

32:15  
So it creates A shortened version of the file name.

32:19  
So let's say we have a file name, which is called the quick brown fox.

32:24  
It's definitely more than 8 characters, right?

32:26  
So the name of the file will be the quick something one.

32:31  
This is, I think, tilled this is never mind and Fox like the end because the last three characters may hold the extension.

32:43  
So that's why the three, three last characters are copied here.

32:48  
And this is like a short version, short name of this file.

32:53  
If someone needs to for whatever reason, access this file system from ASDOS or some very old operating system that doesn't know this extension.

33:04  
And for modern systems, the name is copied into two additional records which are like attached to it, but they don't hold all the data, they just use most of the record to store the letters.

33:18  
And as you can see, the letters are stored 2 bytes each.

33:22  
Here the letters are in normal answer in normal SK1 letter per one byte.

33:27  
Here there is one letter per two bytes because it's stored in UTF 16.

33:35  
And yeah, the Quick Braun dot FO jump X and then there is a 0 as a Terminator of the string and the remaining positions are padded with this value.

33:55  
OK, so this is how we store longer values.

33:57  
What can you notice on this is that there are some additional information like the metadata, create date, create time.

34:05  
Yeah, the metadata of the file are stored directly within this entry.

34:11  
And most importantly, these two items are the most important ones.

34:21  
First cluster.

34:22  
This is the index to the table where the data are.

34:26  
There is some table on the FAT, I will get to that.

34:28  
So this is where the first data block of the file is stored.

34:34  
This is very important so we can find where the data of this file actually are.

34:38  
So this is like the translation we were using with the inode.

34:43  
So similarly we got the first cluster index.

34:46  
So translation to some number where we should find the data and also the file size.

34:51  
So we know how many bytes we can read from this file.

34:56  
OK, this is just an example here, a very old example because that is not that much use anymore, but still it's valid because you get some glimpses and some notions of how this stuff works.

35:07  
OK, so this was FAT.

35:10  
Let's move on.

35:15  
File system or files will be nothing without some file storage.

35:19  
So they need some stuff someplace where they can be stored normally.

35:25  
Usually a file system resides stored on some traditional file system storage, like a secondary or external memory hard drive, SSD drive.

35:37  
By external, I mean it can be an SSD drive connected via USB or USB stick, for instance, or SD card.

35:44  
That's that's pretty much all the same.

35:48  
By the way, you can also have a file system which is directly in the main memory.

35:53  
Yeah, Why the hell we would do that?

35:55  
Because I told you previously that one of the main reasons why we're creating the files is that the application can store them and they will be persisted.

36:04  
They will remain there.

36:05  
They will survive a power outage, they will survive a reboot of the system.

36:09  
But definitely RAM will be erased, will be lost when we reboot or when we lose the power.

36:16  
So why?

36:16  
Why the hell we will store our data, store our files within the RAM?

36:22  
Would anyone venture?

36:23  
I guess, Yeah.

36:35  
I mean, do you understand this?

36:36  
Can you give me an example of this?

36:41  
Yeah, I know you can read.

36:43  
This wasn't test of reading.

36:44  
I I was like, what?

36:46  
What the hell are temporary files?

36:48  
But therefore what?

36:49  
What's the purpose of this?

36:57  
Yeah, but examples, examples, examples, examples.

37:01  
Are you using temporary files right now?

37:04  
There are many people with laptops.

37:06  
Are you using temporary files right now?

37:12  
Maybe, yeah.

37:13  
On the other hand, cookies for websites are very likely stored in the persistent store, unless you're in in the anonymous \*\*\*\*\* mode.

37:23  
So unless you're doing something very naughty on that laptop, it's unlikely that your cookies are stored in a temporary file because you usually want the cookies to survive.

37:32  
If you close the browser and then reopen it, the cookies need to be still there.

37:37  
So cookies are not a good example.

37:40  
When you're installing something, when you're downloading something, when you're unzipping something, you need to extra file where you have a zip archive and you need to take something out of it.

37:49  
You need to just take a look at it.

37:51  
So you just temporarily unzip it to another file, take a look at the file and you just throw it away.

37:57  
So for that, you don't need the persistence, right?

38:00  
You can do this in the memory.

38:02  
And why we are doing this?

38:03  
Because for many applications it's easier to still use the same API for files, even for temporary files, not to allocate their own memory.

38:14  
They used to use some API for files, so they used also for temporary files.

38:19  
OK, now the possibility is network storage.

38:21  
You probably already have seen it, because today many people have their files stored somewhere in clouds, right?

38:27  
If you already used some clusters in Linux classes, I'm not sure whether they do that.

38:32  
Then you probably accessed your home.

38:35  
Actually, you're using them in the labs.

38:37  
If you log in in the labs, your home directory is on some shared network storage.

38:45  
And if you, it doesn't matter on which PC you log in, right, you will still have your home there because it's stored somewhere on the server and all the data are transferred there and back as needed because it's a network storage.

39:01  
And also there are some virtual file systems.

39:03  
So basically many operating systems, Linux especially uses file system extraction, file extractions to create artificial files which are not actually persisted anywhere.

39:15  
But they are attached to drivers, they are attached to pieces of kernel and they provide additional functionality.

39:22  
For instance, by reading or writing these files, you can get some functionality, you can get some information, you can perform some settings to the system very quick.

39:32  
Examples.

39:33  
Dev null is like the one of the most profound files in the in the Unix imposix file systems.

39:40  
You know what dev null is?

39:42  
Anybody knows?

39:42  
Yeah, you probably heard that in Linux.

39:44  
For those of you who are not not nodding, just a quick reminder, this is a file.

39:49  
This is a dumpster.

39:50  
If you need to dump anything anywhere and you need to erase it, you you just dump it to dev null.

39:57  
So all the right operations in devnul are simply ignored by the system.

40:01  
It's just like a eraser.

40:03  
And also if you're trying to read from devnul, it will serve you zeros.

40:08  
So if you need a safe and sound source of pure zeros, you can just read devnul and you will get as plenty as you'd like.

40:18  
Similarly, random will give you random numbers.

40:20  
So if you want some randomized numbers quickly, so you just read as many as you need from your random, it will serve you some random garbage.

40:30  
Another example, if you read this file, you will get a text formatted description of your CPUs.

40:37  
Just try it.

40:38  
Just try to get this file and you will get some basic information about the CPU course of the system you are presently logged into.

40:47  
OK, so files.

40:51  
Another topic which needs to be filled into this puzzle are file links.

40:57  
Actually we have two types of file links.

40:59  
The first one is hard links.

41:01  
Hard links are somewhat like file entries which are duplicated.

41:04  
So hard links from the perspective of the file system aren't even links in this in the simplest terms, it's just duplicating entries for one file block or one.

41:20  
I note I will explain on the whiteboard.

41:23  
It would be easier.

41:25  
Let me just, let's just say you create a file somewhere in the file, somewhere in some directory, it doesn't matter.

41:41  
And this entry translates this file to an inote.

41:49  
Hey, this jokes, let's say 42 because 42 is a really good number for an inote and I don't know, I don't want to give you 2 long numbers.

42:00  
So there is an idote structure where all the metadata are and also the inode holds some references to the data blocks so we can read the file.

42:13  
We'll get to the structure later.

42:14  
So basically we can through this entry we will get the inode and through the inode here we directly have the metadata and through the inode we can get to the data blocks of the file.

42:26  
Easy.

42:27  
But I can do this.

42:28  
I can create a.

42:40  
I can create another file, another entry in another directory which also holds Inote 42.

42:47  
No problem.

42:49  
We can do that because this is just a number, right?

42:52  
So we have two entries pointing to the same inote.

42:58  
Technically and for all instances and purposes, these two files are like the same file.

43:05  
It's like the same file under two different names and that's what's called a hard link.

43:13  
You can hardly.

43:15  
Well that's a good point.

43:16  
It's a hard link because you can hardly distinguish these two files apart and if you write something to 1 file, it will magically automatically appear in the in the other file.

43:28  
Actually there is one way how to find out that there are multiple links pointing to the inode and that's the link count which is stored within the inote structure.

43:43  
You may have noticed it I I've pointed out over here.

43:48  
This is the how many links are pointing to this inote.

43:52  
So if I create this properly, if I create this properly I need to increment this value within this inote structure.

44:01  
Why?

44:02  
Why we need this?

44:11  
Please volume up volume if you want to remove a file.

44:26  
That's right.

44:26  
Yeah, actually it's the other way around.

44:28  
But the the the reason is the same.

44:30  
Yeah.

44:30  
So if I need to delete a file, if I want to delete it, if I'm trying to delete this file and I if I remove the inote and all the data blocks, this entry will become very unhappy.

44:43  
Actually the user will become very unhappy once it tries to access this entry because it will point to a non existing inote and that's something that shouldn't happen.

44:54  
So what happens here is if I try to delete Foo, it won't delete the inote, it will remove this entry and it will decrement this number and the inode and all the data are released once this number reaches 0.

45:11  
OK, so as long as this number is greater than 0, the inode and the data are not removed.

45:18  
So if I have multiple links, deletion of a link doesn't delete the data at all, it just deletes the entry and decrements a number.

45:27  
Only the last deletion deletes the file.

45:30  
That's also why deletion of a file is named unlink in many systems, in Imposix for instance.

45:38  
So you may have noticed that if you try to delete a file using normal operations, low level operations of POSIX, the name the proper name of the procedure is unlink because you are unlinking the inote and the erasure is automatic once the inote reaches 0 links.

45:56  
By the way, important notice, if you perform F open or open doesn't matter on this file.

46:08  
This also increments this link number.

46:12  
So the inote and all the data remains there as long as there is at least one link.

46:20  
And if you even if you delete all the entries, as long as the file remains open, it will be there.

46:27  
This is one of the tricks you can use in programming.

46:30  
If you want a temporary file for writing for some operations, for some temporary data or whatever, and you don't want this file to be accessed by other applications, you just create a temporary file, open it, unlink it, and as long as it remains open, you can use it as a temporary file no problem.

46:53  
But no one else can access it, no one else can seize it.

46:56  
So it's one trick you've just learned if you're dealing with temporary files.

47:02  
OK, so this is hard links.

47:04  
SIM links or soft links are somewhat different.

47:08  
They are basically text files which holds a path to another location.

47:13  
So basically this is like a not transparent link.

47:16  
This is like an explicit link where if you access this link, you need to read the text within the file, which is a normal plain text file, and then use this link as a new location, as a new path where you need to be traversed.

47:29  
So any application that's reading the SIM links need to be aware of them and able to translate this new location to a new path and open another file, etcetera.

47:42  
But usually most of the applications do this transparently and high level functions of the operating system, libraries, et cetera, also can follow similarly transparently.

47:53  
So usually this process is somewhat automized within the libraries.

48:01  
OK, Finally, before we get to some examples, let me summarize what a file system is.

48:06  
So far I've been explaining different parts of the file system, so let me summarize it all together.

48:12  
So basically file system is a set of algorithms, data structures, protocols, everything that's needed to hold, represent, access, store, and everything you need to do with files and directories.

48:27  
So this is like if you sum it all together, you've got the file system.

48:33  
Usually file system is giving you some abstraction, some virtualization, some data structures to abstract files and directories.

48:42  
And by the way, usually operating systems perform abstraction of virtualization of file systems as well.

48:48  
So if you're handling any file system within Linux, it's translated into some virtual file system which kind of encompass all the common things from the file systems.

49:02  
So if you have a different file systems like X or FET, they have some commonalities like they use string file names, they have data creation time for date and time for create.

49:17  
Sorry, time stamp for file creation, that's what I wanted to say.

49:22  
Both of them.

49:23  
So stuff like things like these are common.

49:27  
So even the operating system is trying to virtualize that file system has several responsibilities.

49:34  
First of all, it's responsibility for handling all the naming.

49:39  
So how the directory formatting is done, how the path is are translated into the inote or whatever data block indices are required by the kernel to read the data on the persistent storage.

49:51  
That's all responsibility of the file system.

49:54  
Also all the management of all the data blocks.

49:58  
So basically file system is operating on a some memory storage, some persistent memory storage.

50:06  
So it requires operations like read block number 321, write block number 826, etcetera, but that's it.

50:16  
Everything else is managed within the file system.

50:18  
So any additional files, data structures, any additional bitmaps or linked lists or whatever is needed to maintain which blocks are not allocated, which blocks are already allocated, etcetera.

50:30  
That's within the preview of the file system.

50:36  
Local file systems, the most common common file systems you can find, are usually stored in a normal hard drives, like magnetic hard drives.

50:47  
Yeah, I doubt that anyone has magnetic hard drive here.

50:50  
It would be doubtful because if you have new enough laptops, you're probably using SSD hard drives, but also some removal media, the SD cards I mentioned before, etcetera.

51:01  
Network file systems are somewhat different.

51:03  
They are usually designed for the, for the perspective, for the, for the usage over a network.

51:09  
And usually network has much higher latency than the persistent storage on your local computer.

51:16  
So these have some additional features.

51:18  
For instance, NFS is designed so that you don't have to open files within within the file system itself.

51:27  
So you just providing the some general identification for the file every time you need to access it.

51:34  
So you can access multiple files simultaneously, it doesn't matter how many files you're trying to open etcetera.

51:40  
Also if some, if the connection is interrupted for a short period of time, it doesn't matter because the NFS will just simply recover from from that.

51:48  
So it's a very, very powerful system on Windows, usually using Samba, SMB.

51:53  
This is also, but SMB is somewhat more clumsy because it's basically like an RPC over basic normal file system API.

52:02  
So it's somewhat different.

52:05  
But these are just examples of file system you can see and you can study if you like.

52:11  
I will give you 2 examples in a closer detail.

52:14  
The first one is FAT.

52:15  
You have already seen the FAT directory entry.

52:18  
That's a different part.

52:20  
So let me show you how FAT manages the files data.

52:23  
That's probably the most interesting part of FAT file system.

52:27  
By the way, knowing this plus knowing some details like how the boot record and the initial structure of the hard drive of the persistent storage is laid down is enough for you to be able to write anything, any driver, any, any piece of software that would directly utilize FAT.

52:50  
Side Story, I've when I was younger, I've accidentally erased one of my hard drives.

52:57  
It was like I was doing a complete reinstallation of my, of my PC and I've accidentally plugged as I was shuffling my hard drives.

53:06  
Yeah, I was really physically shuffling my hard drives because hard drives back then were quite small.

53:10  
So I need three of them to manage all my data.

53:12  
So we're shuffling them and I accidentally connect one hard drive to another location.

53:17  
So it appeared under a different letter in my Windows system and I accidentally erased it.

53:22  
Fortunately, it was filled with the data right before that, so it was like all the data are were stored very regularly on it.

53:32  
And I was using FAT file system.

53:34  
So I need to spend like a week writing my own software to carefully extract all the data.

53:41  
Because technically when you perform quick format, you don't erase all the data, you erase only the metadata and the remaining data are still there.

53:50  
So I need to reconstruct it.

53:51  
The FAT table I'm going to show you right now from glimpses of the data on the hard drive.

53:57  
And it was possible because it's a very simple file system.

54:01  
OK, how it work basically besides the boot record, which is kind of boring, there are there is something called FET table.

54:10  
This is the most important part of the FET file system.

54:13  
It holds all the indices of all the blocks where they are stored.

54:17  
So the data there are two copies of the of the FET table.

54:21  
So if one of the copies gets corrupted, you still have the second copy, it's just for the safety.

54:26  
And after that, there is the root directory.

54:28  
So the root directory is basically a file located on a fixed position right after the fad tables.

54:35  
And after that there are data blocks.

54:37  
So these data blocks are data blocks of all the files.

54:41  
And each fad table is basically a index table which holds one number for each data blocks in this area.

54:49  
So every data block in here has a corresponding number in the fad table.

54:55  
That's it.

54:57  
OK, it's.

54:57  
So it's very simple, for instance, to detect which blocks are empty.

55:01  
Empty blocks got zero over there.

55:05  
So if you just scan for all zero blocks, you will find all the empty space of your hard drive.

55:10  
Very simple.

55:11  
And then again, we need to somehow reconstitute the data of each file.

55:17  
And remember, file is basically a data stream.

55:20  
So the file starts on an index which is stored within the directory entry.

55:26  
Let me just scroll back.

55:28  
This is the number.

55:30  
OK, so we take this number which points us to the first location within the Fed table and it's also the index of the first block.

55:40  
We need to read from here, right?

55:44  
And if the file spans over multiple blocks, which usually does because these blocks are usually something like 512 bytes wide.

55:52  
Sometimes actually on the smallest devices, it was like 512 bytes wide on a larger, on a larger devices, they can be quite large.

56:01  
They can be like 1632 kilobytes wide.

56:04  
It depends on your on your device and but, but that's not important.

56:09  
Important is that these blocks are stored as a linked list and this linked list is stored within the indices of the fat.

56:17  
So basically if I have a three as the first number of the first block, I need to read block number three from here and then I need to take a look at position 3 in the fat table where I find the index of the next block.

56:33  
So then I need to read block 10 from the data and I also take a peek at the position 10 in the fat table where I will get the number 15.

56:43  
OK, so the next block is 15.

56:45  
So I read block 15 from this this area and then I go to position 15 -, 14, etcetera, etcetera.

56:52  
Once I've reached -1 and -1 is like a marker, then the the, the link this ended.

56:59  
So, so the position 14, the block 14 is the last block that I will be actually reading from the from the data area.

57:08  
And it's like a termination marker in in the Fed table.

57:10  
So I will have to read no more.

57:13  
OK, by the way, I can somehow determine the end stop also from this information, right?

57:21  
I have the file size, so why it's?

57:23  
Why it's redundant?

57:24  
Why is there file size and always also there is this this marker which terminates the the the link chain?

57:34  
Why I need both of the?

57:36  
Do I need both of them?

57:45  
We sorry, we need.

57:53  
Yeah, that's one possibility.

57:55  
But even if I don't have to look at all the blocks, I need to look at all the positions in the fat table.

58:00  
But if I want to skip first three blocks, I can.

58:03  
I don't have to read them from here, I just need to read the fat table.

58:07  
And actually the fat table is usually cached in the memory, so that's not the main reason I can do that.

58:13  
Even Even so, remember that these numbers represent whole data blocks, which can be 10s of kilobytes long in the worst case.

58:27  
And usually files are not that well aligned.

58:32  
If you have a text file, it can easily have like 42 bytes no problem, why not?

58:39  
And if you just want to read the 42 bytes and the granularity is like in kilobytes, then we need this additional information like how many bytes we want to read from the last block, right?

58:51  
In this case, the first block, because 42 is too, too small a number.

58:55  
But usually the size of the file isn't aligned to the size of the blocks.

59:00  
So you need this additional value like where the where the file actually ends within the last block.

59:08  
OK, another example is X2.

59:13  
X2 is a file system which is being, well actually X2 itself isn't any more used.

59:19  
I think we are using X4 usually or some more elaborate systems.

59:24  
But X4 is still being used on Linux.

59:27  
And X4 is basically X3 with journal and X3 is just an extension of X2.

59:32  
So if I show you X2, you will be pretty much in the picture.

59:36  
You will be pretty much you will get the fundamentals of this system covered.

59:43  
We don't need to get into the details how it changes in X3 and X4.

59:48  
So it uses inotes.

59:51  
We already know that you may have noticed on the previous examples.

59:55  
So the inote is a structure, it's called index node and that's a structure which holds the metadata and also it holds some higher or it's like it embeds some.

1:00:09  
Data structure which allows us to identify the data blocks on the on the drive.

1:00:14  
So it also somehow identifies which data blocks we need to read when we need to access the file.

1:00:22  
And directory is very simple, unlike.

1:00:24  
In fact directory is very very simple table where the names are translated into the inode numbers.

1:00:31  
That's it.

1:00:32  
OK, let me give you a better a better picture because this is much more readable than the text.

1:00:39  
So first of all, if we have a file system, the X2 file system on the hard drive, it usually is divided into block groups.

1:00:48  
So for easier management and for let's say better memory locality, it uses something called block groups to aggregate the data more closer together within the hard drive.

1:01:01  
And also these block groups are easier to manage.

1:01:05  
So they are not that big.

1:01:06  
They don't usually spend terabytes.

1:01:09  
They usually spend, I don't know well, in the past it was like 10s of megabytes.

1:01:14  
Now it's maybe more, but they are not that big.

1:01:17  
And these have a very fine structure.

1:01:20  
First of all, there's a super block and a descriptor.

1:01:23  
These two are like a regular data structures that are always there.

1:01:28  
And by the way, super block and the descriptor are present in all the blocks.

1:01:33  
So super block describes, it's like a structure that describes the entire file system, important information, meta information regarding how large this file system is, how large the blocks are, how many block groups are there, etcetera.

1:01:47  
So all this information is there in the Super block, and this super block is duplicated in every block group.

1:01:54  
So if the part of the hard drive gets damaged, you can still identify the rest of the data because the Super block won't be lost.

1:02:04  
Similarly, the descriptor will describe how large is the structure.

1:02:09  
So since all the block groups are the same, should be the same.

1:02:13  
Then the descriptor should be the same or almost the same and afterwards there are two bitmaps data bitmap and inode bitmap which identify which position in these two tables.

1:02:25  
In inode table and in the data blocks area are occupied which are vacant so which you can use and which are already taken and these two are the most.

1:02:37  
The data blocks are the same as in the case of FAT.

1:02:41  
Here are the real data airlines stored one after another.

1:02:45  
And how we find out which data blocks are used by which file?

1:02:49  
Well, that's the part of the inote structure besides the metadata like the timestamps.

1:02:56  
The inote holds this sequence, this table with block indices.

1:03:04  
So each line here is a number of a block, right?

1:03:09  
And these block numbers are divided in a strange tree like structure.

1:03:15  
The 1st 11 are direct links, so these are direct block links.

1:03:23  
So if you have a file which takes only up to sorry 12, it's from zero.

1:03:30  
My mistake.

1:03:31  
So first 12 blocks can be accessed directly.

1:03:37  
So small files use only one level of indirection in the inode table.

1:03:42  
In the inode structure, you can immediately get the indices of the block.

1:03:46  
So you can immediately read the blocks.

1:03:49  
If the file gets larger, we add another level of indirection.

1:03:53  
So the block 12th, 13th block has one level of indirection, so there is another block.

1:04:03  
And this block is a data block which is filled with links.

1:04:07  
So this has the same size as these blocks.

1:04:10  
All the blocks have the same size.

1:04:12  
But this one is special because it's not, it's not filled with the data, it's filled with references.

1:04:17  
So it's a sequence.

1:04:19  
It's a table also filled with block numbers and subsequent after the 12 blocks.

1:04:26  
So this should be named block 13 actually.

1:04:30  
So block 13 etcetera are over here.

1:04:33  
So you need one level of indirection more you need to read this block and then you can read all the other remaining data box.

1:04:41  
And if it gets even larger you get 2 level in direction.

1:04:46  
So this is a block which points to blocks and three level of indirection.

1:04:51  
So it's like a small files can be accessed quite quickly, quite directly, larger files slightly slower.

1:04:59  
And if you get very large files then this tree gets bulky.

1:05:07  
So this was an example how X2 works.

1:05:11  
By the way, this and this should be might be on the test.

1:05:16  
So it's like a good, good idea to know approximately how this works.

1:05:21  
Because if you don't know how to find which blocks on the FAT table or which blocks in the in the X2 needs to be accessed when you are given a file, then you might have a problem.

1:05:32  
A few more words regarding the storage.

1:05:36  
Yeah, this is like, by the way, everybody know how magnetic hard drive still looks like.

1:05:42  
Yeah.

1:05:44  
So if you know that, then you know that there is a spindle.

1:05:49  
And on this spindle there are multiple platters.

1:05:51  
These platters are made from some.

1:05:54  
I'm not actually a metallurgist, so I don't know exactly what they are made of, but it's a magnetic material and these heads which are placed on an arm, can perform reads and writes using some electromagnetic operations.

1:06:10  
Actually nowadays it's very, very complicated because there is also some laser involved that preheats the the platter.

1:06:18  
It's very complicated from the technological point of view.

1:06:21  
But, but from the, let's say abstractive perspective of our of us programmers, it's very simple.

1:06:26  
You need to place the arm to the right position so it's above the correct track and then you can read or write this track or a segment of this track which are called sectors.

1:06:37  
That's pretty much it.

1:06:39  
The trouble is that when you when you want to access something, you first need to put this arm into the correct position and then you need to wait for the data to rotate towards you.

1:06:50  
The rotation time of this of this whole of all the platters actually is somewhere in this range normally.

1:06:59  
In my old days all the hard drives were using seven 7200 RPM.

1:07:05  
Sometimes the nowadays actually we are trying to be green.

1:07:09  
So it's better to lower this speed a little to 4 or 5400 or something about it because it it consumes less energy.

1:07:17  
It doesn't take too much energy to rotate them.

1:07:19  
But high performance hard drives can go either up to 10K or 15 KRPMRPM rotations per minute, right?

1:07:30  
15 KRPM is fast, right?

1:07:34  
It's 5015 thousand per minute.

1:07:39  
So if you just divide it by 60 right, you got you got the picture.

1:07:44  
By the way, this is like we cannot speed this up much further.

1:07:49  
There were some experiments with filling the hard drive with additional some some helium or something because the trouble is that the head itself is hovering like 5 nanometers nanometers above the platter and that's quite close.

1:08:07  
It's very damn close to be accurate.

1:08:09  
So there are some very specific hydro dynamic properties being observed because as the platter rotates it creates like a small wind right above it and the head itself has some hydro dynamic shape.

1:08:23  
So it basically floats on the wind, which is created by rotating platter.

1:08:29  
Don't ask me, I'm not a physicist.

1:08:31  
But somehow miraculously it works.

1:08:33  
But if you increase the speed, it, it stops working because the the the wind starts to be more turbulent or something and the the heads are starting starting to be shaky.

1:08:43  
So we cannot speed this too much.

1:08:47  
We cannot crank this up much further, but this is basically how hard drive works.

1:08:54  
By the way, if nobody has seen hard drive live, this is magnetic hard drive.

1:08:58  
If you haven't noticed, it's just slightly older than you might be used to, but it's still magnetic hard drive.

1:09:05  
This is, by the way, I think this, Yeah, 10 megabytes.

1:09:09  
So it's huge, right?

1:09:13  
OK, magnetic hard drives, as you sorry, as you witnessed here, these are problems.

1:09:22  
Both the movement and the rotation speed are problems because they are slowing things down.

1:09:26  
By the way, do you know how much does it take within the speeds if you before the data, before the whole track can be read, before the data rotates towards you, roughly what order is it?

1:09:40  
Seconds, microseconds, nanoseconds, picoseconds, milliseconds, whatever.

1:09:49  
Yeah, it's units of milliseconds.

1:09:51  
So it's like, I don't know, 5 milliseconds, 3 milliseconds.

1:09:55  
I think this is 3.1 of them is 3.6 milliseconds.

1:09:58  
I don't remember, but it it's somewhere in this order.

1:10:01  
So it's milliseconds.

1:10:03  
How many instructions can you crunch on your Pentium?

1:10:05  
Oh, sorry, on your on your CPU within 3 milliseconds, right.

1:10:12  
Millions at least.

1:10:13  
So yeah.

1:10:14  
And this is the same adjusting the position of the head, it usually also takes at least milliseconds, sometimes even longer if the path is longer, sometimes even longer.

1:10:24  
So yeah, it's slow.

1:10:27  
We are.

1:10:28  
There were some algorithms that tried to speed things up.

1:10:32  
One way was how to speed things up was to create better scheduling algorithms so that the head doesn't have to move that much or we trying to re pre read the data all the time.

1:10:44  
So basically nowadays when the data needs to be, when a track needs to be read, the, the hard drive itself always reading the whole track, not just waiting for one sector, pushing the whole track into the buffer just to be sure because maybe the next sector will be needed shortly.

1:11:00  
So it's like there are many optimizations in place.

1:11:03  
I would like to show you a few algorithms that were used in in the past days.

1:11:07  
Nowadays they are not that important because we are switching to SSD.

1:11:10  
But still they are there.

1:11:11  
And some of them are interesting from other perspectives.

1:11:14  
So first of all, if you if you handle all the requests on your hard drive, Yeah.

1:11:18  
By the way, this this line represents the radius of the of the platter.

1:11:23  
So this is like a centre and this is like the edge of the platter, OK.

1:11:29  
And these stops represent the requests for tracks which tracks needs to be read or written.

1:11:37  
OK.

1:11:37  
So if we just simply process them first come first serves, it might create a very unoptimized access because for instance, some of these may be far and you need to move reposition the the heads too far from from each each request.

1:11:54  
Yeah.

1:11:56  
But on the other hand, there are no additional, let's say unfairness is added by this algorithm.

1:12:04  
So everybody gets its turn.

1:12:06  
The requests are served in the order they were made, which is the fairest from the perspective of the applications, etcetera.

1:12:16  
Another possibility would be, yeah, we can, we can minimize the seek time, right.

1:12:20  
So let's just say, OK, we, we take the shortest seek time first.

1:12:25  
Let's assume that we already have all the requests in the queue.

1:12:27  
If you don't have the queue, then it doesn't matter, right?

1:12:31  
But if you have the queue, we can we can optimize and reorder the requests so that the the based on the initial position, the short request, the nearest request will be served next, which minimizes the movement of the heads significantly.

1:12:48  
Unfortunately, it's very easy to create starvation because it's possible to keep the request on one side of the platter, and if there is another request on another side, it might wait a long time before actually the hats will get to that.

1:13:05  
So it's not a perfect algorithm.

1:13:07  
Another algorithm is well known as an elevator algorithm.

1:13:11  
Actually this algorithm is implemented usually in large buildings.

1:13:17  
Actually a version of this algorithm to be accurate to collect passengers into the elevators because it would be a bad idea if any passenger could move the elevator to any direction, right?

1:13:29  
Usually if a passenger arrives and wants to go down and then another passenger arrives and wants to go up, it would be a bad idea if the second passenger can change the direction of the elevator, right.

1:13:45  
So what what it does it, it keeps, it keeps like a pointer which direction we are moving and it keeps the direction as long as there are requests in the direction.

1:13:56  
So basically it it creates, it orders these requests into two parts, one within first direction.

1:14:03  
And once we reach the end of the disk, we will start returning.

1:14:07  
And once we reach the center, we will start returning.

1:14:11  
So this reordering is somewhat better because it kind of collects the request as it goes.

1:14:16  
On the other hand, sometimes it's not perfect.

1:14:19  
As you can see, some, sometimes there are long gaps between actually, especially when you flip the side, flip the direction you're moving.

1:14:31  
By the way, it's here.

1:14:35  
Yeah, there is a version of this of this algorithm called Silcroscan, which is like the elevator algorithm, but when it reaches the edge of the disk, it just zooms back to the center and then just stops from the bottom up.

1:14:54  
Why?

1:14:54  
Because it's sometimes it can be.

1:14:57  
This operation can be performed faster than it would appear.

1:15:01  
There might be a special, let's say operation for the heads to just quickly flip from one side to another side and hit some cushion there or something and then start always seeking precisely from the center to the edge or vice versa.

1:15:18  
So sometimes it creates more uniform.

1:15:21  
Also, it can create more uniform time to compare to the elevator algorithm.

1:15:26  
There are some variations on the theme.

1:15:28  
For instance, you can you can have a look and see look which is analogic, which are two analogical algorithms.

1:15:34  
But they try to avoid the edges of the of the platter because both edges, both the center and the edge are somewhat more difficult to handle.

1:15:43  
On the other hand, it creates additional starvations or it creates additional areas which cannot be used regularly.

1:15:50  
So it's not not that much used, OK, this was just to give you a glimpse that there is some, some there are some difficulties going on within, underneath within the hardware.

1:16:02  
And another thing you should be aware of and that's more interesting because it's much more modern is how solid-state drive works because SSDs are still being widely used and they will be probably widely used in the foreseeable future.

1:16:15  
All your laptops right here probably have SSDs, right?

1:16:19  
So this is to give you just the notion how this piece of hardware is constructed.

1:16:27  
So first of all, SSDs are made of memory, which is not that dissimilar to RAM.

1:16:33  
So it's made of some transistors and these transistors are different to regular RAM.

1:16:39  
So they can keep their own charge.

1:16:41  
They somehow can stay in the the last locked state.

1:16:44  
Don't ask me how, I'm not not this is beyond my knowledge.

1:16:48  
So somehow if you set the transistor to one position, you can read it many times without resetting it.

1:16:56  
That's important.

1:16:58  
Unlike DRAM, which needs to be recharged periodically because otherwise it loses its, its coherence, it loses its, its charge, these transistors will keep stuck until you apply extra force to reset them to, to change them.

1:17:13  
And the trouble with that is that if you perform well, not, not the the reads also, but mainly if you perform writes, if you're trying to change the, the state of a transistor, it, it, it gets damaged, it, it's slight, it gets slight small damage.

1:17:29  
And unfortunately this damage is kind of cumulative.

1:17:32  
So you cannot change the state of all these transistors too many times.

1:17:38  
Basically each transistor can be reset like thousands or 10s thousands of times, something like that.

1:17:44  
So not too much, right?

1:17:48  
This means that if you have like this time, this amount of erase cycles, then the mean time before failures is still would be still high if we apply some additional techniques.

1:18:04  
So if we don't damage one particular transistor too much, but if we try to even the damage among all the transistors, then it it still can create decent hard drives and it actually does because we are all using them.

1:18:18  
So we, we are not troubled by this fact.

1:18:22  
You are living in a in a harmony, being oblivious to this impending doom of your hard drives of your data.

1:18:30  
But it will happen.

1:18:31  
Be sure of that.

1:18:33  
So how it's organized, it's organized in two level hierarchy.

1:18:37  
It's organized in blocks and the blocks comprise of pages.

1:18:41  
A page is usually some kilobytes wide, so pages can be some somewhere from 2 to 16 kilobytes roughly there.

1:18:49  
And blocks usually have hundreds of pages, like 256 for instance.

1:18:54  
So if you if you just multiply this, then we can follow the conclusion that the blocks usually have roughly megabytes somewhere there.

1:19:03  
OK, so what's the trick?

1:19:07  
The data can be accessed by pages, so one page can be read or written at any given time.

1:19:15  
But by written, since it's named, we can only write zeros.

1:19:22  
We cannot write once, we can only flip 1 to 0, but not vice versa.

1:19:27  
So basically, you can write a page once and then you cannot write it again.

1:19:32  
Because yeah, technically you can perform a very specific update.

1:19:35  
You can apply some additional writes that will flip some more bits from 1 to 0, but that's a very specific operation.

1:19:42  
This is not being widely used, so the only way how to reverse it is by erasure.

1:19:48  
And the erasure procedure flips all the bits back to once again.

1:19:54  
So you can erase a block together.

1:19:57  
And that's the difference.

1:19:59  
You are always reading write per page and erase per block.

1:20:03  
So you can take the entire block of pages and wrestle all the bits up to once so later they can be write again they can be flipped to zeros again.

1:20:14  
That's why it's called NNT.

1:20:17  
It's negative.

1:20:18  
The reset state is to zeros, and it's end because we are ending zeros, we are flipping bits to the zeros when you're writing.

1:20:25  
That's it.

1:20:27  
And the trouble is that especially the erasure procedure needs to apply extra power, and this extra power is what damages the transistors.

1:20:38  
That's why it works so clumsily.

1:20:41  
But if if this is handled properly, it still creates a very decent piece of hardware.

1:20:45  
It creates a very decent persistent storage because it's very fast.

1:20:51  
When you compare this to the magnetic hard drive, there is no rotation part that there are no heads moving.

1:20:57  
Yeah, So it's very fast.

1:20:59  
It's very, very efficient.

1:21:02  
And also there is the power efficiency, because you need to apply extra power only when you're erasing them for reading, you need much less power.

1:21:10  
And yeah, it's much less power than if you're rotating something.

1:21:13  
The rotation part is very, very costly for from this perspective.

1:21:17  
So let me give you some examples how this works.

1:21:22  
Basically, you start with three blocks.

1:21:25  
And by three, I mean these blocks have all the they are erased, so they have all the bits set to 1.

1:21:32  
All these three blocks are ready to receive rights.

1:21:36  
And at the beginning, you just write 4/4 pages to the first block, to the block what's block X.

1:21:44  
So you write ABCD pages to the block X, and yeah, they're there.

1:21:49  
You cannot erase that.

1:21:50  
You cannot overwrite them usually.

1:21:54  
So if you need to update this data, then you need to rewrite it.

1:21:59  
Let's say that we perform some modifications to the file which is stored here.

1:22:04  
So basically we need to store the additional data.

1:22:08  
In this case we are writing additional EFGH pages in here and then we if we perform some modifications to ABCD then we create the new versions of ABCD.

1:22:19  
Over here, it's ABCD with Apostrophe as a new version of these.

1:22:26  
And these blocks are just marked as unused.

1:22:30  
So there is smart controller running all over the place.

1:22:34  
So the smart controller controlling the access of individual blocks.

1:22:39  
And this actually performs.

1:22:40  
This controller actually performs remapping.

1:22:42  
So it remembers that A is no longer here, but A is.

1:22:45  
The new version of a is here.

1:22:46  
So when anybody asks for a, it will be over here.

1:22:50  
OK, basically this controller is very smart.

1:22:53  
It's like a small computer running within your computer, which just controls the hard drive, the SSD drive.

1:23:00  
And now if we want to erase these four blocks because they are no longer used, we don't want them.

1:23:04  
We want to reclaim this memory.

1:23:07  
If we want to reclaim this memory, the only thing we can do is to copy the remaining valid blocks in valid pages, sorry, from this block to another block without the four block 4 pages which were erased.

1:23:24  
And afterwards, we perform erasure procedure on the first block on the block X to set it free altogether.

1:23:33  
Yeah, very clumsy, right?

1:23:35  
This procedure has several disadvantages.

1:23:38  
Most importantly, pages are being rewritten many times during this garbage collection procedure that tries to reclaim the erase blocks, which adds to the cumulative damage which we are doing to the to the SSD.

1:23:54  
So these reclaim operations need to be performed in a smart manner.

1:23:59  
The controller needs to be really smart to do this only when really needed so that these erasures are not very frequent.

1:24:07  
That's one one like an issue that needs to be resolved.

1:24:11  
By the way, this algorithm I was mysteriously talking about is called wear leveling, that the controller is trying to evenly distribute the workload over all the blocks on the on the whole hard drive, on the whole SSD drive so that each block is damaged equally.

1:24:29  
It's not.

1:24:29  
So it not happens that you, you just keep damaging one or two blocks repeatedly many times, but it evenly shares the workload over all the blocks.

1:24:40  
And the second thing that's there is over provisioning.

1:24:43  
If you buy, I don't let's say 256 gigabyte hard drive, SSD drive for your laptop, right, They're actually selling you more.

1:24:53  
They're selling you like 300 gigs.

1:24:55  
You just don't know that because the extra gigabytes are over provisioning.

1:25:01  
So they are hidden from you.

1:25:03  
So when some of these blocks get damaged beyond repair, so when it's erased and it no longer can retain all the bits up to once, so it's marked as damaged forever and then another block from this reserve from this over provisioning area is taken and is replacing the damaged block.

1:25:25  
So you can run longer than expected with your hard drive without noticing that some parts of your hard drive are already damaged beyond repair.

1:25:36  
That's also normal procedure.

1:25:37  
By the way, there are tools like Smart Tools you can apply to check what's the condition of your SSD drive.

1:25:44  
And these Smart Tools usually read how many of the blocks were taken from the over provision over provision part, how many blocks are inadvertently damaged beyond repair.

1:25:57  
So this is like how you can check for the status of your of your SSD drive.

1:26:03  
There are also some operating system solutions which allows us to even the load.

1:26:09  
For instance they are specified specific flash friendly or SSD friendly, SSD friendly file systems that will allows us to write the data on the hard drive in a more friendly manner.

1:26:22  
Basically they usually depend on something which is called a log structure.

1:26:26  
All the operations that are performed on the hard drive are not stored within the files, but they are stored within the log.

1:26:33  
And log is like a very huge continuous file where all the operations are edit appended.

1:26:41  
So it's like trying to evenly override the data of the whole whole file.

1:26:45  
I'm not going into the details, it's it's quite complicated, but just to give you a glimpse how this, how this might work.

1:26:51  
And second thing that's there is some of the operations need to be added.

1:26:55  
This is an example of an operation that was negotiated between the vendors of the hard drives on the SSD drives and the operating system designers.

1:27:03  
And that's the TRIM operation.

1:27:05  
It's a special command which can be used by the operating system to render some blocks invalid.

1:27:11  
So to tell the the SSD that some of the blocks will no longer be used.

1:27:17  
Yeah, because if you're erasing a, a a file, you're not erasing the data.

1:27:25  
If you erase this file, if you erase this inode, you just mark these blocks as reclaimable.

1:27:32  
You just mark these blocks that they should be retaken at some point, right?

1:27:39  
But you're not erasing them unless you're doing some secrets.

1:27:44  
Unless you need to handle some secrets.

1:27:45  
Unless you need to override them with some random data for protection.

1:27:49  
But usually just keep them as they are.

1:27:53  
But the SSD doesn't know that, so the SSD is trying to do its best to keep them alive.

1:27:58  
It will be.

1:27:59  
They will be copied during the garbage collection.

1:28:02  
So the TRIM operation allows the the operating system to announce that some of the blocks will be will no longer be used and the SSD doesn't have to support them anymore.

1:28:12  
So they can be reclaimed as a free space during the next garbage collection session within the SSD.

1:28:19  
OK, I think this is a good place where we can.

1:28:27  
OK, let me finish this slide and then we will take a break.

1:28:30  
OK, so this is like a fast last slide for the for the hard drives and, and file systems and everything.

1:28:37  
So it's like 2 more issues I haven't covered yet.

1:28:40  
You may notice that I'm trying to put a larger piece of puzzle from many, many pieces because we don't have too much time for this.

1:28:46  
So one thing that you may have noticed, you may have known about is hard drive partitioning.

1:28:54  
If you buy a hard drive, solid-state drive, magnetic hard drive, doesn't matter.

1:28:58  
You can actually place multiple file systems on it.

1:29:01  
You probably know that, right?

1:29:03  
I used to do this all the time because if you have a large hard drive, well, it's not that important anymore.

1:29:09  
But in the past, it was very important that you reinstall your Windows every few years.

1:29:14  
If you had a Windows like 98 or Windows 2000, Windows XP as well, it was very prudent to reinstall the Windows every now and then because it got calm after a long time, because all the updates and everything, it got messy.

1:29:28  
So with Windows 10 and 11, it's not that important anymore.

1:29:32  
But nevertheless, I still tend to divide my hard drive to a operating system part where the operating system and the applications are and the data part and the data part can be done easily backed up.

1:29:43  
And I know that if I want to keep all my data, I just keep the data part and that's fine.

1:29:48  
The operating system part isn't that important.

1:29:51  
It can be replaced, it can be reinstalled.

1:29:54  
So you can do that.

1:29:55  
You can just divide a hard drive into multiple partitions and each partition can have its own file system.

1:30:02  
You can even have like a multi boot solution.

1:30:04  
So you can have a partitions for NTFS partitions for Windows and on the same hard drive you can have XT partitions for Linux no problem.

1:30:13  
You just need to adjust the beginning of the hard drive when the master boot record is or the nowadays the GPT record is.

1:30:20  
This is not the large language memory model.

1:30:23  
This is, this acronym is much, much older than large memory models.

1:30:27  
So this is a different GPT.

1:30:30  
It's just a table at the beginning of the hard drive which tells you which partitions are on the hard drive.

1:30:37  
And another thing you can do is the other way around.

1:30:41  
You can master, you can join, you can bundle multiple hard drives together.

1:30:46  
Again, hard drives, SSD doesn't matter and you can create an array of hard drives.

1:30:52  
It's usually called rate redundant array of inexpensive disks.

1:30:57  
Yeah, I used to laugh to this word because when I was younger than the word inexpensive wasn't really applying to me, especially when I was a student.

1:31:04  
So yeah, it depends on how much money you have.

1:31:07  
If you have too much money, then everything is inexpensive too.

1:31:11  
But this is the way how you can provide either illusion that you have a large hard drive, then you can have, because there are always some limitations.

1:31:21  
Right now you can buy, I don't know, 2830 terabytes per hard drive, something like that is maximum, maybe less, maybe more.

1:31:28  
I'm not following the trends too, too rigidly.

1:31:32  
But yeah, if you need large hard drives, you can collect multiple of these hard drives and create the illusion you have a large hard drives.

1:31:40  
That's called J Bot.

1:31:42  
It's not there.

1:31:43  
So it's like concatenation of the hard drives, but also you can connect them.

1:31:47  
You can you can interleave them in a more clever manner.

1:31:50  
For instance, you can create a rate 0, which takes two or more disks and interleaves them per sector or per block of sectors.

1:32:02  
So when you're reading or writing the data, the operations are distributed evenly among two disks.

1:32:09  
So you double the speed of reading enough writing as well.

1:32:14  
If your controller is fast enough, you can really double or multiply.

1:32:18  
If you have multiple disks, you can multiply your operating speed.

1:32:22  
Very useful.

1:32:23  
But then again, you're creating a more dangerous situation because if one of these files, one of these drives fail, you lose everything.

1:32:35  
So it's not always the best option.

1:32:37  
It's a very good option for scratch drives.

1:32:39  
If you really temporary storage for huge scientifical data and you really need to process them fast, that's perfect solution because you don't keep the data forever there.

1:32:49  
You just need them for some computation.

1:32:51  
So it's it's better if you can access them fast.

1:32:54  
The other way around is that you have RAID 1.

1:32:57  
You have two disks and they get mirrored.

1:33:00  
Both disk retain the same data.

1:33:02  
Every write operation is sent to the the same is sent to both drives and both drives perform perform the operation the same.

1:33:10  
Still write 1 can double your speed when reading because when you're reading you can request the data from both drives because they're mirrored, they're the same.

1:33:20  
So you can you can request half of the data from one OneDrive and another half from the second drive.

1:33:24  
So basically you're doubling your throughput for reading and there are more elaborate ways how to create rate.

1:33:30  
For instance, rate 5 creates a block of n + 1 disks, and the N disks are used for data and the plus one disk is for some check sums.

1:33:41  
It's slightly complicated than that, but basically if you lose one of these n + 1 drives, all the data will be still intact.

1:33:50  
So if you like use 4 drives and one extra drive, you have 5 drive.

1:33:55  
So you have the capacity of four and the extra drive will provide the the check sums.

1:34:00  
So if any one of these five drives fail, you still have the data.

1:34:05  
So it's you.

1:34:06  
It's like a safety precaution because sometimes the hard drives fail.

1:34:12  
So that's it.

1:34:13  
Sorry about the sorry about the gap.

1:34:16  
I will suggest we take a 5 minute break.

1:34:18  
We'll suffice.

1:34:19  
So if you need to refresh yourselves, go to the restroom, whatever you need.

1:34:23  
So please do and we will resume in five to 10 minutes based on how fast you will be.

1:34:28  
OK, I realized it would be difficult to sit here for another hour with a hood break.

1:34:34  
So please take a break and we will resume with another topic.

1:34:42  
Hopefully this will also resume the recording and let's switch to another topic which is virtual memory.

1:34:52  
Technically, from the perspective of operating systems, this probably the most important topic because it will give you insights into many things that are happening inside your computer.

1:35:04  
That also includes memory allocation, file operations, etcetera.

1:35:10  
So far we will.

1:35:12  
We were working with the assumption that there is some memory which is given to the process.

1:35:18  
When we were starting the process, I told you about the segments of code, segments of data like constants, etcetera, which are somewhere in a linear memory.

1:35:28  
But I was talking about things like there are multiple processes running together within the same operating system on the same host, and they need to somehow be separated one from another.

1:35:41  
And that's what this old concept, the whole concept is about.

1:35:47  
South.

1:35:47  
Technically, from now on we will distinguish two types of memory.

1:35:51  
Mostly.

1:35:52  
When I was talking about memory, I was talking about virtual memory.

1:35:56  
So that's the memory that applies to regular applications.

1:36:00  
So regular applications are running in virtual memory space.

1:36:05  
They are given the illusion they are living in a dream world.

1:36:08  
They think they have all the memory for themselves.

1:36:12  
They have a huge memory space, 64 bit memory space only for themselves.

1:36:17  
But that's a lie.

1:36:18  
And this lie is made possible by hardware and operating system working together.

1:36:27  
Because the real memory, the actual memory, the DRAM you plug into your CPU to your main board is the physical memory.

1:36:35  
So this is the real memory.

1:36:37  
And there is a mapping.

1:36:38  
There is a translation mechanism that allows translation between regular pointers of regular applications, the virtual pointers into the physical pointers into the real memory.

1:36:49  
So every time CPU accesses memory in the user mode, it always perform this translation.

1:36:57  
So every access to memory goes through this translation mechanism.

1:37:02  
So the CPU basically translates some pointers which are given to it to load instructions, to load data, to store data, whatever into the physical addresses which are then going to the buses, which are then used to access the real memory.

1:37:18  
That's the main concept.

1:37:20  
That's the main notion.

1:37:24  
All the memories, both memories are always one-dimensional.

1:37:28  
Actually there are ways how to ensure how to create multi dimensional virtual memory like segmentation where the addresses are actually 2 dimensional.

1:37:37  
You have two parts of the addresses, but that's not important.

1:37:41  
Physical addresses needs to be always one-dimensional.

1:37:43  
This is just a sequence of bytes concatenated 1 after another.

1:37:48  
There are ways how to create multi dimensional virtual addresses by.

1:37:51  
Usually we are also working with one-dimensional addresses because it's simpler.

1:37:56  
It's easier.

1:37:57  
The tricky part is how to ensure that this translation mechanism is effective and efficient.

1:38:04  
So first effective.

1:38:05  
It needs to really translate all the addresses properly so that each process is shielded from all other processes.

1:38:13  
That's difficult on its own.

1:38:15  
And second, it needs to be damn good efficient, because if it isn't, then it will slow down all the computations significantly.

1:38:24  
So the CPU needs to be doing this transparently and very, very efficiently.

1:38:29  
And one of the most important parts here from the perspective of operating systems is that sometimes this translation, this mapping may not exist sometimes because it's a mistake.

1:38:41  
If you are trying to access memory which wasn't previously allocated, that's something you may have already encountered in your Arduino classes.

1:38:50  
So if you are trying to access memory where you shouldn't be, so you have a wrong pointer by accident or by a bug in your code, it's a fault.

1:39:00  
And this fault will result into termination of your application.

1:39:03  
The operating system will kill you, your application to be exact.

1:39:07  
OK, so the this is what happens and this is handled on the hardware level as an exception.

1:39:13  
So if this mapping doesn't exist, the CPU raises an exception.

1:39:18  
This exception goes into some exception handler which is a part of the operating system.

1:39:23  
This is code provided by the operating system, and this exception handler needs to decide what's to be done with this exception.

1:39:30  
If it's a mistake or if the operating system thing it's a mistake, then this exception will translate into on Linux into signal 11, which is 6F.

1:39:41  
If you heard of it, you may have noticed it.

1:39:43  
You may have encountered it in the record X.

1:39:45  
That's what happens if you are doing something bad with memory.

1:39:49  
So 6F will terminate effectively and effectively your application.

1:39:55  
Sometimes it also may be used to some trickery.

1:40:00  
For instance, if you allocate huge amount of memory, you allocate TB of memory, then Linux says, no problem, here's a TB of memory.

1:40:09  
But it's a trick.

1:40:10  
It's a, it's a ruse.

1:40:11  
If you try to access the memory, then it will slowly associate the memory with real physical memory.

1:40:17  
So it's like it's something like a credit card in a bank, right?

1:40:22  
You theoretically have this huge amount of money on the credit card, but in the end, if you try to draw the money, then we will have to pay the bill.

1:40:30  
So it's the same thing.

1:40:31  
You will get the virtual memory.

1:40:33  
So you will get the illusion that you got the memory, but the memory, the physical memory will be allocated later as you try to access the memory.

1:40:41  
So basically every time you try to access a new part of this allocated memory, new exception will will rise and the operating systems will say, OK, this is not a, it's not a mistake.

1:40:53  
I actually promised this memory to this application.

1:40:56  
So now it's time to make good on my end.

1:40:59  
And I will just allocate physical memory for it and and create recreate this mapping and then restart this instruction so it will properly execute without without exception, without fault.

1:41:11  
OK, so this is like a basic mechanism, the most important mechanism which is there, which you really need to know from the beginning.

1:41:20  
And now.

1:41:21  
Yeah, I forget about this.

1:41:22  
This is the new meme.

1:41:23  
I haven't, I haven't showed you yet, Right?

1:41:26  
So as an example, make sure that everybody understand that because the previous slide was all letters and I was just waving my hands.

1:41:37  
So just let's reabsorb it on a real example with a real picture.

1:41:42  
So this is a virtual address of the process.

1:41:46  
You already recognize that this is the picture I took from the some of the previous classes when we talk about how data are organized within a process.

1:41:53  
So there is a segment where the code is some constant, some data heap, etcetera.

1:41:58  
So each part has some proper location in the virtual address space, usually starting from zero.

1:42:05  
Why not?

1:42:07  
And the memory unit, the memory management unit of the CPU uses some instruction, uses some data, some mapping which was provided by the operating system to translate all the virtual addresses into physical addresses.

1:42:23  
So every time the data are accessed, a constant is loaded from this address.

1:42:29  
This transparently happens within the CPU.

1:42:33  
So actually this part of memory, physical memory is accessed right and returned as the constant.

1:42:39  
OK.

1:42:41  
Furthermore, all these addresses are handle it in the hardware.

1:42:45  
So everything what happens in hardware like caching also works with physical addresses.

1:42:52  
So L1L, 2L3 caches all work with physical addresses.

1:42:59  
Why we are doing this?

1:43:00  
Well, some reasons I've already told you.

1:43:03  
The most important reason, the one I should probably start with is security.

1:43:07  
We need to provide the illusion for each process that it's that the whole system is its own, that the whole memory is its own.

1:43:16  
Well, actually we are technically doing the same thing with CPU using time Multiplex, right?

1:43:20  
We are giving the illusion that the process is alone on the CPU, but actually we are taking it off the CPU once in a while to allow other processes run well.

1:43:30  
With the memory, it's more difficult because we cannot use time multiplexed with memory easily.

1:43:35  
So we are dividing the physical memory into multiple virtual spaces, making sure that each virtual space has its own physical memory.

1:43:45  
One of the reasons, one of the other reasons is it was used in the past.

1:43:50  
We can create an illusion that we are having a huge enormous memory space, petabytes of memory right in 64 bit addressing.

1:44:00  
Yeah, it's an illusion because physically normally you can have up to TB easily if you have enough money, several terabytes, perhaps more is much more difficult.

1:44:10  
Yeah.

1:44:10  
So the physical address space, the virtual address space is much larger than the physical address space.

1:44:18  
Actually, nowadays it's it can go the other way around, because if you have a 32 bit system like AI, 3232 bit implies 32 bit pointers.

1:44:28  
So if you have only 32 bit pointers, your virtual address space is limited to 4 gigabytes of memory, which is not much by today's standards.

1:44:40  
So it's very easy to create a machine on the 32 bit architecture or 32 bit application which can use only four gigabytes of memory.

1:44:48  
But technically the host may have much more.

1:44:51  
My laptop has much more than 4 gigabytes, so no problem today.

1:44:56  
So this is no longer primary reason because usually you are not using petabytes of memory and usually you are satisfied with the amounts you can get.

1:45:04  
And also thanks to the virtualization of this mapping.

1:45:08  
Thanks to this concept we can create additional features easily.

1:45:14  
One of them is memory mapping.

1:45:17  
Create a memory mapping of a file is very easy.

1:45:21  
You just promise that some data of a file will appear somewhere in your virtual address space.

1:45:28  
So again, the operating system just remembers that when anybody asks for this data, it needs to be loaded from this file.

1:45:37  
So the loads themselves are performed using this handler, using the exception.

1:45:43  
So whenever anybody tries to access the data promised by the memory mapping, the data will be provided on demand, on demand by and handled by these exceptions.

1:45:54  
Another thing is we can use it to data sharing.

1:45:58  
We can share data among multiple processes.

1:46:01  
Actually that happens all the time because there are plenty of data in this area which are the same for multiple processes.

1:46:12  
Why?

1:46:12  
Because we have libraries.

1:46:13  
We have low level libraries, operating system libraries, pieces of kernel that needs to be there in every application.

1:46:20  
And it will be very wasteful if we need to load these libraries every time any application wants that.

1:46:27  
For instance, the Lib C library.

1:46:30  
It's not that big, but it's not that small and you are running thousands of processes on your own Linux server.

1:46:37  
If the Lib C needs to be replicated 1000 times in the memory it would be very wasteful.

1:46:42  
But we don't have to do that, We just need to map it properly so it's loaded once in the physical memory and every process gets like a virtual copy of that only using this memory mapping mechanism.

1:46:57  
OK, important concepts.

1:47:04  
Sorry skipped 111 other sentence how to do this.

1:47:09  
This was why.

1:47:10  
So we are now we are switching to how and there are two possibilities how to create this mapping principle.

1:47:18  
The older and I'd say more clumsy way how to do that is segmentation.

1:47:24  
Segmentation is no longer being used in current systems, but it's a good place to start because you can see how to do it more easily.

1:47:32  
So the concept is that the virtual address, the virtual space is divided into logical segments.

1:47:39  
So some like blocks.

1:47:42  
This was originally designed to go 1 to 1 to the concept of work of the division of the process space, because you have one segment for code, another segment for constants, another segment for heap.

1:47:56  
So you have different parts of your application that needs usually needs separate handling.

1:48:02  
For instance, the code needs to be read and executed, right?

1:48:06  
The data needs to be written and not executed, etcetera.

1:48:10  
And the trouble is that you don't have, you don't know how large the code segment will be, how large the data segment will be, how large the heap segment will be.

1:48:20  
So you need to make this segment adjustable.

1:48:23  
Each segment may have its own size.

1:48:25  
That's the main concept of segmenting.

1:48:27  
So each segment have a different size.

1:48:30  
And of course these segments have some identification, for instance, an ID index number.

1:48:39  
So finally the address has two parts, the segment ID, segment number and segment offset, which is how many bytes from the beginning of the segment the data are.

1:48:52  
So basically each segment creates kind of a small virtual subspace of its own.

1:49:00  
Because if you fixed the segment number, then by moving by changing the offset, you can move only within this segment.

1:49:08  
If you have fixed segment number for the code segment, you can only access the data.

1:49:13  
You can only access the instructions in the code segment.

1:49:15  
If you fix it for the heap, you can only access the data on the heap and the translation is provided via something called something which is called segment table.

1:49:26  
It's really like a translation table in memory where all the descriptors of all segments are located and each descriptor provides a base pointer to the physical memory.

1:49:37  
So it provides a base plate for the translation and some additional information, for instance, the length of the the size of the segment.

1:49:44  
That's that's important.

1:49:45  
And also perhaps some additional attributes like whether the segment should be readable, writable, executable, etc.

1:49:54  
And something which is called a segment fault.

1:49:57  
And by the way, this is a term which we are retaining today, even though we are not using any segments anymore within our CPUs because segmentation has been deprecated, it's no longer used.

1:50:10  
But still we keep the name segmentation fault, segment fault when we cannot access the data within appropriate within some segment.

1:50:19  
So if the translation or the validation of these access rights or something fails, then we get this segment fault.

1:50:28  
How this works?

1:50:29  
So we take the virtual address, it has segment part and offset Part 2 parts.

1:50:34  
2 dimensional segment part is basically an index to this table.

1:50:40  
So it's an index of the segment descriptor which is a structure I record in this table.

1:50:46  
So we read this record and we need at least two important information from base physical address pointer.

1:50:54  
So we take a base address.

1:50:56  
This is the beginning of the segment within the physical memory and we need some limitations.

1:51:03  
So we need to remember the length of the segment.

1:51:06  
And the translation is basically an addition.

1:51:09  
We just take this pointer and the offset, we add them together and that's our desired physical address over here.

1:51:18  
Because the offset from the beginning in the virtual address space is the same as the offset in the physical address space.

1:51:23  
That's the amount of byte we are shifting.

1:51:26  
It's it's easy.

1:51:27  
Of course we need to perform some additional checks.

1:51:30  
For instance, we need to perform a check that the offset is smaller than the length.

1:51:35  
Otherwise we can end up in a completely different segment and that would be disastrous.

1:51:40  
So there need to be some additional check that the length is smaller, that the length is greater than the offset.

1:51:49  
So that's how this translation works.

1:51:50  
Is it clear?

1:51:52  
OK, so as you can see it's not very practical because every access to memory will result in two reading this data structure, and even if this data structure were cached somewhere, so we still need to take this base pointer and perform an add operation.

1:52:11  
Add is not that fast, so each memory access instruction results in at least one add and one comparison.

1:52:21  
So we need to compare offset with the length these are.

1:52:24  
These two operations cannot be skipped.

1:52:26  
By the way, do you know how comparison is implemented?

1:52:30  
What is comparison technically?

1:52:36  
Subtraction, which is addition.

1:52:38  
So we're performing 2 additions technically with with some small modifications within the processor.

1:52:45  
That's right.

1:52:48  
That's why segmentation is not that practical.

1:52:51  
And there are, sorry.

1:52:52  
There are a few more additional issues.

1:52:56  
One of them is the different sizes of different segments create fragmentation within the physical memory.

1:53:05  
Imagine that you want to enlarge the heap.

1:53:09  
You need to reallocate the heap because heap is growing and you need to reallocate it.

1:53:13  
You need to enlarge in it.

1:53:14  
But heap is hitting this barrier.

1:53:17  
Yeah, you can.

1:53:17  
You can easily change the length as long as these two rectangles don't overlap, right?

1:53:23  
So we can increase the size of the heap slightly, but once it reaches the border of the constants segment, we are out of luck.

1:53:33  
Well, actually not not entirely.

1:53:35  
We can shuffle these, we can move these segments no problem, but it's tedious, clumsy and slow.

1:53:43  
So yeah, we can deal with fragmentation, but at heavy costs, so it's very impractical.

1:53:49  
This is why another concept was created called paging.

1:53:53  
Paging goes with a different direction than.

1:53:55  
The main difference is that the paging uses pages and frames, so some data blocks of equal size.

1:54:05  
So they have some predefined size, which has to be not maybe, but has to be power of two.

1:54:17  
Yeah, power of two.

1:54:18  
And both virtual and physical address space is divided into these blocks of the same size.

1:54:25  
That's like a must that does the main, main thing you need to need to remember.

1:54:31  
And yeah, just for technical notion, the virtual address space blocks are named pages and the physical address space blocks are named frames.

1:54:41  
But that's just to distinguish whether we are talking about virtual address space or physical address space because they have equal size.

1:54:48  
So one page fits exactly 1 frame exactly.

1:54:54  
And the virtual address space is now 1 dimensional.

1:54:58  
We have just one address.

1:55:00  
Why?

1:55:00  
Because thanks to this condition, we can easily chop off the offset.

1:55:05  
We know what offset part is.

1:55:07  
We know in which block.

1:55:08  
We can easily take the index of the page just by cropping the proper amount of bits from the beginning of the pointer.

1:55:16  
Or we can take the offset part just by cropping this amount of bits, the end amount of bits from the beginning from the end of the pointer.

1:55:23  
So we we don't make this distinction in the virtual address space.

1:55:27  
We just use one linear number.

1:55:30  
And where exactly is the border between offset and the page number?

1:55:36  
That is no concern of programmer.

1:55:39  
Programmer doesn't care, all the pointers are the same.

1:55:44  
OK, now we need a mechanism to translate virtual address into physical addresses when we have this concept.

1:55:52  
So this translation mechanism is usually called page table.

1:55:56  
Well, the simplest mechanism is called page table.

1:55:59  
And again page table is a translation data structure where we have a for each page, we have a, a offset, we have a, no, sorry, we have a base pointer to the physical memory.

1:56:09  
So again it's indexed by page number.

1:56:12  
Similarly, like the segment table was indexed by segments, and each entry contains A-frame number and possibly some attributes, again, like attributes like whether the memory is readable, writable, et cetera.

1:56:27  
Whether the translation is valid.

1:56:29  
That's also a good flag that should be there.

1:56:33  
And that's pretty much it.

1:56:35  
By the way, this table is usually designed so that each entry has the exact amount of bits as the the address size.

1:56:46  
So if we have a 64 bit addressing, each entry in this page table has 64 bits, where the hell we find the extra place for these attributes?

1:57:08  
Actually, this is a trick question which aims to test whether you followed all this information properly.

1:57:17  
That's it.

1:57:23  
If we know this, there is one peculiarity about pointers stored in this page table.

1:57:33  
Physical address pointers stored in this page table.

1:57:38  
What is it?

1:57:47  
Well, we know that all the frames have the size.

1:57:56  
So if we know that every frame has a size which is power of 2, what will What will happen if we try to write this address down in bits?

1:58:14  
Read it in a binary nobody?

1:58:23  
Well, this implies that each page and each frame starts at address which is divisible by this.

1:58:36  
Yes.

1:58:36  
No, this so far, so clear.

1:58:39  
OK, so this this part is clear.

1:58:41  
So if the address is divisible by a power of two 2 to the power of N 2, Yeah, exactly.

1:58:53  
The last N bits will be 0.

1:58:56  
That's what LIMN is about, right?

1:58:59  
So the last N bits will be 0.

1:59:02  
So if you're storing the address into this table, then we have all the 0 bits at our disposal.

1:59:10  
We don't need them.

1:59:11  
We know they're zero, so we can reclaim them for other purposes.

1:59:15  
So there will be at least N lower bits in this entry, free to use for other purposes, for other flags, for instance.

1:59:23  
OK, so let me let me go again step by step how this translation works.

1:59:30  
So we have a virtual address, it's linear from our from the programmer's perspective.

1:59:36  
But during the translation, we need to split it in two parts.

1:59:40  
And we are splitting it at the bit level because we know that N minus and N lower bits belong to the offset.

1:59:50  
Easy.

1:59:51  
It's this division can be done on the linear number just by chopping off the lower end bits.

1:59:57  
Easy peasy.

1:59:58  
And the remaining bits, that's the index to the page.

2:00:03  
That's the index of the page.

2:00:04  
So that's the index to the page table, right?

2:00:09  
So this value P basically you can take this address and shift it by N to the right and you got the index.

2:00:18  
Right, but technically shift isn't a mathematical operation.

2:00:22  
Shift is that you just take the bits.

2:00:25  
If this is a constant, then it's not a mathematical operation.

2:00:28  
You just take the bits and send them over a different set of wires.

2:00:33  
You don't have to perform any mathematical operation, unlike adding.

2:00:36  
So technically you just take the upper bits to index a position in this table, and you take the lower bits and put them aside.

2:00:43  
Here in this table, you read the upper n -, n bits and take them as an address to the physical memory.

2:00:53  
And you append, you override the lower end bits with the offset you stored here.

2:00:59  
So again, there is no mathematical operation.

2:01:01  
You just take these bits and copy them over here, just send them over a different set of wires.

2:01:07  
That's it.

2:01:08  
And this is the physical address.

2:01:10  
Voila, no additional operations.

2:01:13  
So it's much faster than adding.

2:01:15  
There is no adding.

2:01:16  
There's no math involved.

2:01:17  
You're just sending the data.

2:01:19  
Only thing that is involved here is you need to access the memory in this table and read this value from this from this table.

2:01:27  
Yeah, you need to perform that.

2:01:29  
And of course, if there are some significant bits in the lower part which indicate whether the memory can be accessed, whether the record is valid, whether it can be read, executed, whatever you're doing with the memory right now, yeah, then additional operation needs to be performed there.

2:01:43  
But usually this operation will involve checking single bits.

2:01:47  
So it's easy.

2:01:49  
It's easy for the hardware and the CPU to implement how this page table will look like.

2:01:55  
OK, so basically if you want to cover the entire virtual address space for every page in the virtual address space, and there are plenty of them, you will need to create an entry within this page table.

2:02:08  
So each entry will keep the address.

2:02:11  
Some of these entries may be empty.

2:02:13  
There is some flex saying this is just an empty space.

2:02:16  
We are not using this page at this moment.

2:02:21  
Some of them have valid addresses.

2:02:24  
So this page #14 refers to the page table entry #14 there is mapping one to one.

2:02:32  
Page 14 goes to the entry 14 and there is number 456, which is an address, physical address, which points to the frame 456 in the physical memory.

2:02:43  
So this is how the translation works.

2:02:44  
And yeah, some of this, some of this are not or explicitly marked as as invalid.

2:02:50  
So when somebody tries to access this page on this frame, it will blow up in his face because the CPU will arise an exception.

2:03:01  
And then this exception, this page fault will be handled by the will be handled by the operating system.

2:03:07  
So again, this might be a bug.

2:03:09  
So the operating system will kill the application or it might be intentional because the the operating system is doing some magic behind like loading data from memory mapped file, swapping the data to the to the hard drive, providing the allocated memory on demand or something like that.

2:03:30  
OK, a few things we need to spill out and I think yeah, we will do these two slides and we will conclude for today.

2:03:38  
So few more things you need to be we need to spill, spill out right away.

2:03:43  
So we yeah, you probably don't you might have already noticed them, but let me go through them one by one because this is like a difficult stuff for many people.

2:03:53  
So first of all, let's say I have this one level page table you you just seen on the on the previous slides, this one.

2:04:00  
And I have a 32 bit address space, both virtual address space and physical address space to be on the simplest side.

2:04:09  
And I have 4 KB pages, which is kind of a common size 4 pages.

2:04:13  
Normally we are using like 4 KB pages.

2:04:15  
Some architectures use different sizes.

2:04:18  
On IA 32 you can use 4 KB or 4 megabyte pages.

2:04:23  
So it's it's a normal size.

2:04:26  
So we have 12 bits per page per offset, right?

2:04:31  
2 to the power of 12 is 4 kilobytes everybody.

2:04:34  
I I didn't for the time purposes, I didn't ask you that, but I believe everybody sees that.

2:04:40  
So what is the size of the page table entry 11 this.

2:04:47  
What is the size of this box in bytes, bits, whatever?

2:05:07  
I'm waiting.

2:05:09  
I'm not going to make this easy on you, right?

2:05:15  
You got all the information you need.

2:05:16  
You have 32 bit addresses in virtual space, 32 bit addresses in physical space.

2:05:22  
Everything is 32 bits.

2:05:24  
And we have 4 cool pages, so each page requires 12 bits for offset.

2:05:33  
Just to be clear, OK, if I have this page table, how many bits, how many bytes or whatever I need to allocate for each entry, what does this entry hold?

2:05:48  
What is within this box?

2:05:56  
Yes, I'm very glad for that.

2:06:00  
At least somebody's paying attention.

2:06:01  
So thanks for that.

2:06:03  
Just please give a bit more time to the rest of the class.

2:06:08  
I don't want to dampen your spirit.

2:06:10  
I just want everybody to realize that it's a physical address.

2:06:16  
Did you catch that?

2:06:18  
OK, so who catch that can answer me the question.

2:06:21  
How many bytes or bits or whatever?

2:06:23  
What will be the size of one entry in this table of one of these boxes?

2:06:37  
It's one physical address.

2:06:41  
So what is the size of physical address?

2:06:54  
32 bits, That's right.

2:06:57  
Actually I can, I can drop the lower 12 bits because there will be zeros.

2:07:01  
So technically I can go up, go with 20 bits, but it's very impractical because 20 bits is that's not a very good number.

2:07:10  
So usually we will go still with 32 bits, we will just recycle these 12 bits, 4 flags and other stuff.

2:07:16  
But yeah, one entry is 32 bits, 4 bytes.

2:07:20  
So we are moving.

2:07:21  
So what's the size of the whole table?

2:07:24  
How many entries are there in the whole table?

2:07:31  
OK, I will, I will go easy on you.

2:07:32  
How many entries are there?

2:07:37  
You can read How many entries are there?

2:08:04  
Nope.

2:08:05  
I'm not sure how you get to that #128 I understand that perhaps you don't see the detail because it's written in small numbers.

2:08:19  
So you can apply whatever zooming device you have, or you can read the slides from your laptops or whatever, but you will you should be able to deduce it on your own.

2:08:37  
What is the large N?

2:08:38  
Yeah, I realized that I have two NS here.

2:08:41  
So what is the large N?

2:08:45  
There is a large N and small NI realized we are on this for 2 1/2 hours already and yeah, I'm exhausted as well, but bear with me.

2:09:00  
Bear with me.

2:09:00  
Just keep it to one more slide after that and we'll we're done for today.

2:09:06  
Master some, some lasts, emergency results and some power to your brains.

2:09:13  
What was the last what?

2:09:14  
What is the large N here?

2:09:22  
This is the size of the address.

2:09:24  
This is the number of the size of the offset, the number of bits in the offset.

2:09:33  
So I'm relying on you, nobody else.

2:09:40  
Sorry.

2:09:41  
Volume up.

2:09:47  
Thank you very much.

2:09:48  
It's 32 bits for the big N and 12 for the for the small 1.

2:09:52  
So can you, can you please finalise it with this hint?

2:09:55  
So it's N to the power, 2 to the power of large N minus small N 22 to the power of 20 is roughly million.

2:10:08  
Yeah, it's like a mega, roughly 1,000,000.

2:10:11  
So we have 1,000,000 entries in here.

2:10:13  
So what is the size of this table?

2:10:20  
You just need to multiply it with the entry size.

2:10:24  
That's it.

2:10:28  
Someone from this side of the audience please.

2:10:31  
I haven't heard anything from this part.

2:10:35  
Anyone.

2:10:38  
We have 1,000,001 mega to be precise, 2 to the power of 20, 4 million.

2:10:47  
That's right.

2:10:48  
4 mega, 4 megabytes actually of memory we need for this page table.

2:10:53  
So it's large, it's huge, not that huge, it's affordable, but it's quite large for managing 4 gigabytes of memory, right?

2:11:01  
And the most important part is we don't need it all for normal applications.

2:11:07  
If you run an application that consumes fewer megabytes of memory, that's for a simple command line tool, that's more than enough, then you will allocate another 4 megabytes just to keep this page table.

2:11:19  
That's a lot.

2:11:20  
So what can we do with that?

2:11:22  
We can go we can go into two level 2 level indexing.

2:11:29  
So basically we split the table into, we split the address into half.

2:11:36  
Well we can split it into multiple chunks, but in this case half would be perfect.

2:11:41  
So it so happens that with four KB pages, half of this of this page table index is perfect.

2:11:50  
If this is 20 bytes, then 1/2 is 20 divide by divided by two is 2 10.

2:12:02  
Thank you.

2:12:03  
So trying trying to reach out with basic arithmetics.

2:12:08  
So we divide it into 10 bits and 10 bits and still the offset remains 12 bits as it as it was before.

2:12:14  
OK, so because we still have 4 KB pages, so the offset doesn't change, we're just splitting the index of the page.

2:12:22  
That's it.

2:12:23  
And so it so happens that once the index is only 10 bits, we don't need MB of entries, or 4 megabyte mega of entries, 4 megabytes of entries.

2:12:37  
We need only 1K of entry, so 4 kilobytes of entries, right?

2:12:43  
But then again we need 2 levels because we need the same for the second level.

2:12:48  
So basically the first level will hold the pointers to the second level page tables.

2:12:55  
It's like you are splitting up an array into two level tree structure.

2:13:00  
The first node is only one and it will be always there and there will be up to 1024 second level nodes.

2:13:10  
And it so happens that these tables are 4 kilobytes long, so they will fit perfectly into our pages.

2:13:19  
That's just a coincidence, but actually it's a happy 1.

2:13:22  
So technically we are allocating pages for these tables.

2:13:27  
This page needs to be always in the memory and we need some beginning point, some starting pointer where the root is.

2:13:35  
And this pointer is actually stored in some special hidden register of the CPU.

2:13:39  
So there is some system register that cannot be accessed by regular applications, only by the operating system, where the operating system sets the pointer to the first level of the page tables the physical address, right?

2:13:54  
This holds physical address of this second level page table where the second index is applied.

2:14:00  
And finally, here is the address of our final page table where the actual data are.

2:14:06  
So we apply offset to it and we can read or write the data over here.

2:14:11  
What's the benefit of this?

2:14:12  
The main benefit is that not all these tables are there.

2:14:17  
This table needs to be there all the time, but we can save a lot of memory because many of these tables may be missing, may not be there simply, and if they are not there, we are saving memory.

2:14:27  
So if we have a very small application, we can allocate only a few of these second level page tables.

2:14:34  
So we allocate only a few pages to manage the whole translation process.

2:14:40  
I will quickly go through this again next week because this is something worth remembering.

2:14:46  
Just a few more notions.

2:14:49  
You probably notice that each translation of the memory requires several accesses to memory as well, which is very, very unfortunate.

2:14:58  
So what happens here is that CPUs usually employ additional caches.

2:15:04  
They have special names, TLB translation, Lucaside buffers, and this is a very specific cache designed only for memory translation.

2:15:12  
So this cache is like regular caches associated, but it's very small, usually like 10s or up to 100 entries, something like that.

2:15:21  
And it has, it only provides translation between virtual addresses or page tables, page page addresses to physical addresses.

2:15:31  
So it's like a very special cache which can be accessed very fast.

2:15:34  
So not always we need to perform this entire entire lookup, but this lookup is performed only once for every page that's being accessed.

2:15:44  
And then the address is stored into this cache.

2:15:47  
So next time you're accessing data on this page, it's quickly resolved through the cache.

2:15:52  
It doesn't have to go through the entire lookup process again.

2:15:56  
That's the point.

2:15:57  
And by the way, some, some processors like MIPS, which you already seen use something which is called zero level page tables, which means it has only the TLB.

2:16:08  
So it only uses this cache and the operating system can insert entries into this cache manually.

2:16:14  
So it's like another way how to deal with memory translation.

2:16:18  
OK, sorry, I'm I'm slightly over the expected amount of time.

2:16:23  
Are there any questions for this, for today, for this topic before we conclude for today?

2:16:29  
I hope next week we'll be roughly on the same schedule.

2:16:32  
So we will again have the extended lecture, but I hope we will manage to do it somewhere within this amount of time, so roughly around six.

2:16:40  
We hopefully conclude any questions for today?

2:16:44  
Not yet.

2:16:45  
OK.

2:16:46  
I suggest next week at the end, if time permits, I would like to dedicate a few minutes 1st to go through the testing process to give you some glimpses about how the example look like.

2:17:00  
We will go through some sample sample tests cases, sample test questions.

2:17:06  
And also it would be a good idea to, let's say, go through the entire course before attending tomorrow, next week's class.

2:17:14  
And if you have any specific questions regarding any topic we covered during the the semester, then it will be a good time to pose these questions.

2:17:23  
So I'm not pushing anyone, but it might not be a bad idea to just rehearse everything, refresh your memory or go through the slides before next lecture.

2:17:34  
So if you have any questions, you can easily ask them during the session.

2:17:39  
Thank you for your attendance for today and hopefully see you next week.