Week 7 Lecture 2

Yes those things are still on the test yes I have limited time right alright so today we're gonna talk about file system IO and then we're gonna cut out hopefully early yes I'm gonna talk about the homework before we go yes I wanna make sure of two things actually you know what let's do it now.

Really simple if we remember how a file system works how does the file system work we talked about it right directories have directories on them and then we go on and on and on those directory files are actually list of inodes and pass right pretty simple if you remember those high level concepts it's very very simple if you try to forgive those high level concepts gets harder pretty quickly OK but finally it was actually easy now this is where your programming skills come into play recursion is your friend not your enemy yes recursion use recursion right because directories are defined recursively on purpose yes however we have some bad news so you remember I told you or not maybe that told you maybe your instructor told you a million years ago anything you do or you be a recursion you can do bigger iteration right this is kind of like a form of doing something over yes there are so many ways to look forever when you do recursion with file systems so be aware of them I'm gonna point out as we go along but you sort of try to internalize those yes your next homework doesn't have file system I have stuff right the one after that does however right from that use it right like it's not there's something anyway we talked about file metadata stat permissions and ownership right because now we're changing all these things we discussed directory including entries creation removal opening directory streams and reading directories exciting we're gonna use links we're talking about links including the hard links and symbolic links symbolic links and their system calls for the love of God avoid this like the plague and it's it's not actually that complicated really isn't symbolic links and hard links are actually simple they really are however symbolic links if you follow them you have to follow a whole big set of rules otherwise you're gonna real trouble really quickly yes because essentially follow symbolic link and then all of a sudden you might be looking at the exact same directory that you already looked through but with different paths and therefore it looks different roll right so it's really important to avoid those those situations where you can then we'll talk about copying and moving files we have some really simple ways to do it recall that each file is referenced by anode which is referenced by inode these are internal structures organized by the file system by Unix but I noticed both the physical object located on the disk and the logical entity complete with the data structure in the kernel right the data structure and kernel actually references the item you knew about file tables and how that works yeah I know stores the metadata associated with the file who owns the file for permissions the file has public file is all those things are actually managed right all of those are part of the data structure you can take the I know number using the I flag with the LS gonna have you ever feel like you need to Oh my God you don't need to yes we're we're not in this isn't a Unix class so as a result of that I don't really care if you can do these things right if you want to if you're looking for wearing out why this file I looked at this file four times or something go ahead yes go ahead do that but we don't need to do that provides a family of functions to end the metadata on the file and I already forgot two of them why should happen right no seriously so you have two that are important stat maps that start maps that you can see one takes a path and that structure structure from the file system this is that F stat takes that file descriptor right it's the other type of open so you open the file and then you get it stacked you'll likely use the first one more and then the second one based on what we did the third one I believe and I have to look this up for the forgot third one I believe looks at links yes however don't care right because I'm generally avoiding things because I don't want to do too much with links yes alright so this is gonna get weird so this is your stat structure OK the first thing is the device ID device ID is not gonna be SDA FDA or something like that right it's not it's actually a number associated with the file system so that number could be a variety of things to actually get that actually out you have to issue a couple commands right it's not fun to find it but it is findable right but what we typically do with the device ID is we don't actually look for certain device ID we actually compare it right is this like is this file the same as this file as long as they are they're on different devices obviously they're on different things right so we use that to compare it more than we actually use it to find or identify something right then I know numbers there then there's St. mode or permissions the number of hard links is there the user ID of the owner of the group ID and device ID bad bad bad news OK so couple things we don't know yet so we can't do right now but those user ID group ID guess what they're literally the number associated with the ID internally kernel the Unix kernel I don't know why that's happening sorry internally the Unix kernel uses an ID number associated with a user right doesn't use the username and uses 1000 one 101 whatever as a result we don't have the command right now we don't we will soon but we don't now right now have you a way to actually get the username from UID we will very soon if you have our dev it's a device ID it's usually some sort of special file and then St. size is total size in bytes then you have the block size this is the block size for the file system that happens to reside on does not the block size of the actual file right that's the block size yes and then the number of blocks that are gonna be on and then the most importantly for us is access time modification time and status change time and you know what access time is last time somebody opened it up modification time pretty easy what is that you're too good for this the last one status change time is a little weird the file isn't changed because that would be the modified time hasn't been accessed the access time status change means anything else somebody creates a hard link to the file somebody changes the permissions that's what we're talking about another one that was gonna make I didn't get there yet 8 \* C time like excuse me 8 \* m time and C time again we have pretty ways to print those out and we'll see those later if it becomes valuable for us to do that I will I will do some it's just simple command it's really just include time and then like it's like it's so simple it's like print time or something like I just don't remember off my head but it's a pretty simple command just to print the time but these time T things are not just simple things so here's a static sample we have a command line starts that SB we're gonna create a variable named SB then we're gonna go ahead and stat RV1 which is what did you watch the recordings for Monday hey does everybody never know jeopardy right did you watch video

Anyway RV one is the second argument going into the because argv 0 is what it's the the actual executed command right so our V1 is actually the first argument passed to the command line program right the first argument is actually the name of the executed file remember OK so anyway Rd. one is whatever passed to the command line and we pass SB usually by it's pointer sorry address and then we get the step back and then we get S equals LD bytes you pass the fargate RV and you pass and then you get the St. size so you're gonna actually print out the size invites sexy right if you look at the stack and you really look at it what is there that's absolutely useful not much the modify times are important who owns it may be important if you're looking through if you're trying to create a fine like program like Unix has a fine like program there's some things you could do but print out right directly print out briefly there's not much there except for size right because if you try to pronounce C time and time what's called it just a big number I think it's the number of seconds since the epoch right you know this computer science right it's 11 11 70 killing these anyway yeah I should know that right it's the 1170 anyway whatever these are the things but like user ID yeah you can print that out sure you can print that out and it would print out a number 1001 one thousand whatever was that actually useful not really right and so yeah size is the really one that we use now if you were trying to create a find like program or find program on Unix by the way you can look up the command find actually allows you to find files by name by you know if you wanna look at certain size size you can use size if you want to look at a certain file type can use certain file type we haven't figured that out yet we don't know file type yet if you wanna look at the modified time you can do that right you can compare modify time with another one whatever find to let you use a massive number of things to look for a group of files yes you didn't even file things by 4 great but individually like printing things out there's not much you can do OK so same thing made art anyway this is start again and it's gonna print the file type this is not trivial unfortunately sucks asss actually in my opinion I think that if I had the power if I had the power right I would go like this word in the massive sword or whatever power greaseball would go back and put a file type in here right because that is exceedingly useful 99% of the time you look through directory entries or stat files what are you looking for is this a directory or is this a regular file right pretty simple things you need to know about files that you're looking for you can't do that directly but inside and we'll try to inside the node inside the Meadow on the mode stat mode if you end it with SIFNT right it's the file format right you can actually find out that's where the number exists for you to see if it's a if it's a character device node or a block device node this means it's a character device file or a block device file and you're like wait what you can actually look at devices from the stack right which is why you get those but then if directory the below and if fifo we don't care about here but that's a first in first out it's a pipe pipe you know Mario goes in one side comes out the other side if word so I was gonna tell you joke about Mario goes in the pipe and then Francis piece goes in the pipe right and outcomes Mario right first in first out and then outcomes Peach and then outcomes that's different than and then you got a regular file right and then you got a regular file so this is 190% of what you do when you look through these things and you're checking for files and for directories I believe there's another way to do it but this is the most one of the most common ways you can also look at sockets and if you don't know what the bottom you don't know something I know right now it just doesn't happen it could be something weird but these are all in the book by the way so make sure you check that out stack can be used to obtain permissions but two other calls can actually be used to step seven we don't set much or do anything with the actual stack we just made this the status of the file wanna change permissions we have two mechanisms do it one takes about the script or one does not exciting ownership same 1L1 is the one that uses I'm farting great party I think that's the one that does links that challenge takes Obama scriptor Chong does not right it takes a path an owner and a group again those are ID's those are not usernames and yes later on right now we don't know but we will later on when we go over users and permission from those type of things we can actually take a username convert it to an ID and take an ID and convert it to a username we just don't have that ability yet because we haven't gotten to that point we do that OK yes we're finally at the point where we can talk about alright can you next directories the simple cloud containing the list of directory entries each is an I know name map is great we already know that every directory has two directors and you have to be make sure that you are filtering for both of these right why let's talk about it so right here means this directory so suppose I'm going through directories and every time I find a directory I call another method for directory I get right here and I call the director what happened looks forever repeatedly keeps looking at right here oops the other one is dot dot right my parent this is just as bad if indirectly bad so I send it off to the other method the other method goes through my parent finds my directory and eventually it looks right that one's actually suckier to find that will say the dot when you ever have you ever accidentally feed the dot in it just loops forever and you see the same directory being reviewed right but when you have that room it actually takes a minute you see all sorts of random garbage and you're like how is this that's what that is so every process has a current working directory current working directory is wherever the actual path of this system is actually running every program you run will have a current working directory now this refers to the starting point for resolving relative path names and that's the only reason that's important only reason current working directory is important it's important for the following reason when you're opening files and doing things with files if you use a local relative path name guess what yes if you're in the right direction but if you use that relative path and you're not in the right directory yes the process can actually obtain and change its but I want to point out a couple other things one of the things we get from directory files and we need to be very very cautious about this is when we actually move through the directory we get a list of file names not paths OK So what directory might contain Bob Mary and sue I don't think we have Bob barriers yeah we definitely don't OK Bob Marian suit as an example it does not contain C: slash windows slash blah blah blah blah blah right what happens if I need to call the command on this file in this directory I need to rebuild the path or at least remember the path in order to do that if I need that that's important OK obtaining or changing right we can get current working directory we just need to send a string buffer and size right you have to have a Max size bag done change directory you just give it a path 90% of the time when we're changing the working directory we get it we get it working directly we do something and then we go back to where we were 99% now well 95% of the time yes you're moving the directory to do some job you move through the directory to do some job and then what do you do you go back we go into a new directory or whatever yes this is the case we often use the FDA's this more efficient but it doesn't creating directories yeah make directory permissions you have mode so the mode is ended with a umask so confusing but trust me just make a directory and then LSL and see what it does and then you know how to manipulate from there this is useful for me because I know what I'm looking at yes unless you're knee deep in the woods and system administration in Unix this is a little confusing for most people what I do most of the time is I just create a file figure out what I have and then figure out where to go from there why it's OK screw up right learning the process yes remove and directories RM dir give it a path there's no recursive version Unix won't let you delete something with something in there So what do you need to do director note that you cause you already talked today I'm gonna give you your break today although that seems weird to say out loud sorry didn't mean it that way start however I only because you know this can't get any closer Riley so we can't remove the directory if you have stuff in it so but I wanna remove the directory So what do I need to do yeah I wanna get rid of this trash bin but I can't because it's got stuff in it I can't move it So what do I need to do get rid of this stuff in it then I can move it yes that's it now how do we do that work I know I can't help myself I realize how dumb it was but then like you know when you're when you're live you're lik content is actually really simple removee yeah it's live right we will live on give me any worse so i'm sniffling a little bit because i had type

A file everything gets easier if you forget that you're like wait why does this work yes so door opened her given her character name when you do that you have to include Durant dot H is guess what you gonna be reading talk about later our director scheme is basically a file scripter with some metadata contain the FDA contents we have a director a series of directory entries we read the directory and it gives us the Durant right just yes every time what did you ray of direct gives you all the durants I like singing Durant it's fun yes just imagine the \*\*\*\* coming out of my mouth in 10 minutes right anyway but no it's just an array of directory entries and again I know number if the offsets and the next one we don't care about that one of the caveats of directory entries is directory entries are in fact name that is variable a length of record a deep type all those things this is a variable size struct as a result of that we need an offset having said that do we actually do anything with the offset no right that's pretty cool so I don't know about you guys but I like to know that I don't I can ignore something right the record I don't think I really care about that either to be honest with you but D type so remember with their stats way back here we need to do this really crazy thing to figure out directory type like figure out what kind of thing it is my holy \*\*\*\* right it's complicated now this type tells you what it is super simple yes and then of course you have the don't forget the file name is not the full photo path of the file is literally file very important for you to if you need to open another program that requires the full path like you're gonna open a file as an example put the path back on there yes otherwise it's confusing OK oh closing directory stream you just close it like file links alright hate this a link is essentially an A for an item right we create a more than one link to the same I note these are references to hardlines Hardwick is literally nothing special there was a time when Unix we shared things on servers by giving two people the same name and giving them permissions to do what they needed to do and that's how they shared about exciting I mean that's literally how that works yes however we don't really use that too often yes the system still periodically a couple of servers will actually modify the some of the same files right they might be hard another type of link is a file system mapping but a high level pointer is interpreted runtime these are symbolic these are your problem why hard links they don't really cause endless loops they might cause you to get the same file right you look violent twice oops symbolic links are actually links to actual tasks so you can actually create analyst loops here very quickly and easily OK they may point to non-existent files which are dangling symlinks these are essentially like shortcuts on your desktop that you would no longer go anywhere you deleted the where it goes that's what they are best description for them they allow usually what we do them for is just like with sharing files we share you a symlink and you're like you suppose I'm hosting a file hosting a web server and so I want to share access to something so I give you a link a symbolic link in your directory that links to where you are that gives you access to that directory but nothing else yes that's kind of what you're trying to do but you can essentially choose to follow this is how you create one that's how you create a similar don't let it care alright alright what I want to talk about really quickly I haven't done it yet is we wanna avoid some links as much as possible it should be easily avoidable yes but if you're looking in the API you say hey this one follows some links and this one doesn't always choose the one that does not follow so much yes I might have you looking at var logs as an example just to look for whatever how many lines are in the how many lines are in the var log files or whatever yes give you some system administration tasks to do because Malik lake happens to be in there and you follow it could be bouncing all over the file system yes so always always at least in this class there are times in your real life practice where you may need to do that but you have to follow a whole another set of rules about following right alright remove we don't need to do any of those copying there's no system call for copies most of the thing do is open open old file open the new file copy the file and close yes we don't really need to do that there's a copy command but you had to do it it's pretty easy food moving you can rename it give it give it the old name and give it a new name moving is different coping does what takes the bites over there moving it no longer has the bold names old thing no longer exist it's a new thing now we discussed file system mail including metadata stat permissions and ownership count the number of text files in the system pretty quickly right you do that super quickly right then all of a sudden you forget to look for dot or dot dot you're like what the hell did I just do yes this program supposed to be super easy the difference is actually pretty minor it's the fact that recursion is not people don't practice recursion and the way they the way you may have practice iteration and so it's a little bit harder to see a little bit harder to understand but like once you've had some time with it it's 90% of the time it's really simple we're gonna implement something like that next class we'll see how many lines are in barlog literally usually but we'll be doing it across the entire file system right any log file any text file regular file with the name login we knew whatever we want remember we have the ability to do contains we have string and the string we have the ability to manage string manipulate string tokenize all those things we have the ability to do just about anything across the entire file system how many atoms exist in Texas across the entire file system take a look that's not a real example it's not but I do want to talk about a couple things So what you don't know and you don't really need to care about because I don't really care that much but there's also been magic files in the Unix file system that actually stores certain types of files so you can actually look for PNG files certain types of executable files by looking at the first case number to be able to know quickly how to move through the entire file system looking for something specific it's useful also for those of you that are gonna be looking for buyers removers or you're trying to build that sort of thing looking at my patterns the way you do it how that works we're gonna do some stupid examples but real life that's that's literally how that works alright so we talked about the green directory entries creation removal opening directory schemes and reading 100% of the time you're going to be reading and doing many moving around the file system rarely are we gonna actually be creating when we get to servers we'll switch gears a little bit have a directory called a working directory so if the file system if the server needs to store temporary files to anything we'll do so in that working directory it's the only place they'll have permission to the directory doesn't exist or allow them to create an example yes everything that's pretty common so we might do that but do we need to maybe just maybe depends on the character server when we talk about different types of servers do different types of things that's really more distributed system class but for instance if you're writing database log you might actually database logs you might store them on a daily basis that allows for easier rotation of log files that's an example and so in a folder for that day you put all the changes for that day as an example and that makes for easier rotation of log files and you don't know what that is but that's a backup mechanism so anyway we're talking about links including heart lakes and symbolic lakes and what are we gonna do with links Phil avoid avoid symbolic links yeah and then anytime you look at a command and it says that you this will follow symbolic links what do we do yes those L's I kept saying sorry why isn't that going back even no earlier I was talking about these commands that start with L like L chone those type of things believe those involved avoid them we don't need to do anything there that whole set of rules there's a whole set of rules for traversing symbolic links that you have to be careful because the symbolic link could actually put you you could be looking through a directory follow symbolic link to actually somehow put you in the same directory all sorts of things can go really wrong really quickly so it's important not to all of those just gonna make your life easier and of course some of you just don't care if your life on