October 18, 2023

Process Management

--Exam next week online. Nov 1 to 4, recording class. –

The homework we do after the exam will not include it in the drop date guidance, right? So we won't have it back in time for that. Second thing is as I just talked about. As a result, I'm gonna try to grade your exams and second exams, second homework, three, all those things prior to me leaving this. But that's important because the drop date is actually November 2nd, right? So if you wanna make a decision about not being in the class anymore, that's your last day to do it. Process management, though a good benefit. Sounds like IT stuff. But we finally get into four, right? 4K, then having to wait. And we're gonna talk about zombies. Yes, we're talking about killing processes, because that's the term we use. It sounds murderous, blah blah blah. It's good day for Halloween, right? Maybe I should. But the reality is that's just the terminology we use to kill processes and so forth. So I calculated the extra credit for the thing. I did not do anything with it. I will not do anything with it till probably next week or the week after. Redesignation thing happening and I have to do that by next Wednesday, so I'm basically putting everything on hold. To take care of that. So, but you should see that before the dropping once The other thing. Yeah, and I mentioned homework four is coming out today. Homework three should be due tonight. Again, no solution will be posted, but you're gonna have massive penalty. That's OK Also make sure you do your own work. You can earn a zero. I know this is gonna sound crazy, but I just wanna put this out there. It is possible for you not to do well, of course, yes. So do your best on the exams and so forth study. Like I said, you know, my exams may not be the easiest thing in the world, but if you study, I'm sure you do. Well, yes. But I feel like sometimes people think. That it's like there's a lot of alternative ways to end that. Alright, let's talk about process fundamentals. Process fundamentals include process concepts. The number one thing with all the things we talked about from this point to like the like signals, timing, what's called all those constraints, timers and so forth, you have an identification mechanism, have a way of getting the process abstraction and getting details about the processing. What does that actually mean? Process abstraction. Remember, everything is a file. As a result of that, we should have a way of opening it up and see what the memory allocation is to that process and so forth. So we'll talk about how we do that. We'll talk about process components, which goes into a process API for using processes and executing new ones. Whole lot of way to do things. However, be aware some of them do some funny things. So be aware I explained the process termination concepts right. Terminate processes, the Terminator. It's not the 90s, right? Some of you were just being born in the 90s? Not in this case, right? We're gonna make sure that process goes away for good. Best managing users and groups of processes, including queries and change. Now we actually have to talk about users because when we have a process, processors are allowed to do whatever processes are allowed to do. So now we have the first time in which we might get a failure because. You may not have permissions to do something that is a problem, so we'll talk about that as well. But we'll also talk about Ch mod and so forth. We do some demos as well. The most important, not the most important, but another important very related concept is the dammit concept. This is the. End result of what? We'll be working towards an application that's gonna run this server in the background and it's gonna be a background service. And its core implementation of literacy, right? So all we're gonna learn is how to create a gaming process, how to disconnect it from process groups, and so forth so it won't be terminated by default when something else is terminated. Alright, processes are object, code and execution. These are active running programs now. The actual quote, We don't manipulate, right? That's not something we're worried about. We're worried about making the code not run anymore, right? We're starting code, so it executes. Process also has various resources which are all managed by the kernel. Every process gets its own view of the operating system and it can't interfere with others. That's the goal So the current manager using buses. Descriptive but. Process time. You have memory, you have space available to you. The kernel manages all these resources. Process is really what we call it. Virtualization abstract. The reason we do this is we create a virtualization of the computer. The reason we do it is so that when we have executing processes, they cannot interfere with each other because if they interfere with each other. All hell breaks loose, right? That's a huge problem. We want to keep all the bulls running in the same direction, doing their work or doing whatever they gotta do without smashing into each other and stopping themselves from going. OK, we don't want to process using up the entire memory space. We don't wanna process or using the entire CPU time. Where something else can't run, the kernel provides them with this process. Process believes it's operating on its own, all by itself, all out here by itself. Right. And that's actually really important concept and. Linux is the basic basis model for a lot of what Windows does now because of this virtualization abstract, because of how well it actually worked. But the core thing we're seeing too is Android and so forth also use this model, and they use it in an even stronger way. Which. Turn into separate environment. But it's a pretty important concept of binaries, compiled executable code that is dormant when you talk about a binary in Unix. It refers to a program that's executed, not executing. Executing a process is a running program. This includes the binary itself. Itself has to be loaded in memory. That is not. Trivial that has to be loaded in memory. And it's placed in the. It's place in where the running code is actually happening. Which line is executing, as an example, is maintained throughout, right? So that's something that's happened. Memory, kernel resources, security context, and one more threads may be associated with. Threats, by the way, too, the unit of activity within size processor. It has its own virtualized processor which includes a stack, a processor state such as. Wait. That's not what I think it is. That's the IP address. It's something else. Instruction pointer that stands for, yes, I'm sorry. So earlier I was talking about having to maintain that process and where the executed is, that's actually maintained in the thread. But the same basic premise is happening. We have to have the structure point, we have to have registers associated with, we have to have processing resources available, right? Each process is represented by unique identifier referred to the PID, guaranteed to be unique in any single point in time. Kernel allocates them in strict, strictly linear fashion until it reaches its Max. 32768 will not reuse it until it wraps around. This generally makes it easy for them to not give out. Yes, Now, having said that, Linux can actually do whatever the hell it wants. Remember they told you about the non standard thing. This is Debbie. OK they try to do this. Right, they all have the main uniqueness, but they can do whatever they want, yes? If you maintain a table is relatively easy. Once the process is finished executing, to hand out that PID again, right? It's just relatively easy if you maintain a resource that does that, for example. Do you need to do that? No. But generally speaking you're guaranteed to be unique. So that's the important part here. The other process, which is the process the kernel runs from, there are no other running processes, PID zero. This part first process started is called the net PID one. This actually boots the system and launches all the login. So you probably don't wanna go kill. Dash 9/1. Right, that's the command to actually kill a process. But if you kill them primary process, it actually shuts the system down. Because when you think about it, I wanted you to think about it this way. From units perspective the process transforms tree. The processes form a tree, so when it starts everything else so all those other processes are connected to that. So if I kill the net. And kill the root. What do you think happens to the tree? Yes. So be cautious about that. That's very important concept we're gonna come back to in a minute. We're not system administrators, so we're not worried about killing Annette. Really. You could try it, by the way. Go ahead. Virtual machines are built for this. Just make sure you save a snapshot before you do it. Yes, but they're built for this. They're built to be destroyed and rebuilt. What? It won't actually hurt anything. You know your computer might be unusable. You said power off and then reboots up and it's fine. But we're not saying administration right, but this concept when we have shells and we have those type of things that are user uses, those open processes and those processes that open to those shells belong to that shell. So when that shell closes. Showing those processes, and that's important. The process that follows new process is known as the parent. Remember I told you that word for killing terminated process is literally killing. We're gonna be killing parents and and this is the terminology we use. I'm sorry. OK, the process as far as the new parent new process is known as the parent new process is called the child. Every process is spawned from another process. So every process has a parent. That parent is referred to as the PID. Ha ha ha. I didn't know if you were gonna fall. Thank you for falling. They trade not. They trade not. We all have that inner child in this. It's OK. Each process is owned by user group. There's no situation where you have a process that's not owned by user group. What if I wanna have something owned by nobody? Nobody's active user, and within the Linux operating system typically it's an active user, just has no rights. Has no home directory, has no files. That is still a user, yes. And that is how we maintain those things. Everybody has to have a user and has to have a group. This is what controls access rights for that process. A process does things as a result of doing things that may. Open files and make close files. All those things. The process is where we actually determine if you can open or not. Not what we've done so far. The process is where we determine, right, Whoever owns. That is how we do it, Which is why we haven't really messed around with using groups up to this point. Each child process inherits the user or group of their parents. Pretty simple. It's a very, very simple process. It's really easy. If I execute a program, very likely I am the parent, yes. You go, wait, what? How is that? Well, remember a shell? When you log into the system, you actually log into a shell. That allows you to manipulate the system that show processes known by you. As a result, any process is spawned from that shell. Are also home, right? OK, each process is also part of a process group. Now this is a little bit weird so. So process groups are groups of processes organized for some reason. It's not really hard to understand that part. But and this is also here, but the real thing, the reason we do it is so when we have to say signals to a process or a group of processes, we can do it very easily. Signals like terminate, signals like go, signals like stop, signals like continue. All those signals. We don't know about signals yet. We will soon, but that's why we have a process group. So when we group them, we're grouping them generally administratively, right? So actual purpose of grouping. But we'll learn about later when we have damons. This is a big problem because when we actually want to close our shell, we don't want the Damon to close as well. Yes, we want the damage of stay open. But as soon as I close my shell, it since terminates everybody. So we have to get out of that process group, learn more about that little bit. Alright, so get the PID and the PID. It's a long number, not very interesting. Get PID, get PID right with their ID right a new process. There are two ways of doing it. Exact and exact family. I hate the exact family. It does some weird stuff, but it is useful. It's not dumb. I've used it personally, put it that exact loads of binary program which replaces the parent. There is no more parent child. Yes. The parent dies off. And the child is now the process. Imagine it you morphing, it's not quite the same, but imagine you're morphing parent to child. That's what's happening with exactly. So if you wish to maintain control or wish to manipulate is probably not what you wanna do. This is what we refer to as executing the program. There's no single function there, actually a bunch of. That's why I said the exact family. And you're gonna love this. They do a bunch of different stuff. Use this one in this situation. Pain in the ass. It's not like there's one that's good. It's good for all of them. Yes, there are actually lots of reasons to do that. And then four. Windows before kids, right? Fork in the road is what we're thinking of here. It creates a near duplicate of the parent initially and then you could do whatever the hell you want. But. Starts off with the same code base. Or they can go and do this line of execution, but the child remains in the parent remains and does it's there. Yes, it's a near identical group of the parent, but we can check the PID. PID to know which ones? The child a parent. The child can do one thing, the parent can do another thing, and you have two forms of execution Ave. two processes. This is different from exact I'm gonna keep repeating. I know this is a horse lying dead in front of me. I got my stick. I'm still gonna beat it. OK exact, but you have to have a pot child and parent exactly is not for you. OK, if you need show on the parent, that's 4K with child with four with the exact. The child becomes the parent or the parent becomes the child. Start the other way around, right? There is no parent anymore. With fork we have a parent, parent child. Executive primarily used to initiate you run another program, right? Could be a system program or something, but we'll see. Alright, I love this Word, so Excel Exec L sorry, not Excel reading. Constant car art. And,... boom. Is very attic. Variable number of. Parameters, yes. It's. It's a random number of variables. Why did he do this? It's very important that it don't. Doesn't use a pointer array. Right, you just have a set of arguments and then you have the last one in with null. Why? Because we wanna be able to issue options to the command so it's gonna execute. Path, that's gonna be the the path of the command, and then we're gonna send arguments, and it could be a number of arguments, we're just gonna put them in. Composite real list we're just gonna end up. Here's an example. Then slash buy. And I'm gonna stand by slash home student dot homework dot text and then I'm gonna end with no, of course. And what's that gonna do? It's gonna open by with. Slash home slash student dot text. Give me any command on the system happened. This one happens to be 55 by the way, if you're not familiar with it. Yeah. Send this new one. The old school doesn't fancy stuff. What's the difference? Opening singles. So if you have signals for the parrot that are pending. Before the exact call. Bye, Bye. Go on. Any payments, pending signals, you're waiting for something, you're doing something, you're doing this, doing that. They're all gone. This table is the process of catching and return to their default behavior. Learn about catching signals. We can catch signals and do something different with them. That all reference to the default all that goes away any any memory locks that you might have. Bye bye all locks. Most of our attributes are returned to defaults. Anything related to cross memory space is cleared. It's like a brand new process and the best way to think about this is this. Imagine for a second exec algo wanted just showed you. Literally starts a new process from the command line. It stops and starts new process from the command line. While this will be true all the stuff on the top. Dimensions and new processes. Anything that exists in user space, anything floating in memory, anything variables, anything on the processor or anything in the registers, all that stuff. So what's the same? PID. Paranoid. If you had a priority established, that would be the same and whoever the owner, user, group, that's it. Quite literally the only thing that you get if you do exact that stays the same is the PID of paradigm, right? And of course the priority, but. Priorities are suggestions in Unix. What's the suggestion mean? They're very familiar with the suggestion. You should eat breakfast in the morning. Yes. Did you all eat breakfast this morning? Some of you did. Some of you just like Nope. This is suggestion do it or you cannot do it. The fact is, it's sometimes followed, sometimes not. So we knew. Try to enforce priority. You need to actually do some certain things to do some checking to see if it's actually working OK, so don't worry about it too much. If exit kills the parent and makes the child the parent. How does it keep the process hiding? It's like, OK, so the is, The process still exists, but it's. OK. So it keeps it's it's original ID I mean. Except in the process. Like, not. Actually, that's not actually what happens. Does that make sense? Yeah, it more. Exact happens. This is what's gonna process so that the new process keeps the process ID from the from the old oh. Rest of the exec. But I will say. So much more sense to me. So much more sense. If you pass an array. As the arguments more sense than to do that variable argument thing. Exact LP? Doesn't matter. Oh exactly allow you to add the environment. And LP allows you to use a file rather than a path. PS Saying bottom this fourth one, I think 123 Yeah P is the same thing. Open the file BE as the file, but it also has environment. So if you wanna pass environment variables or whatever, you can do that. Yay. Alright, good news, bad news. We don't use this that often practice. And why we use it all the time but? We use it for fairly specific purpose and we rarely use it directly. OK, So what do you mean we rarely use it? Just like this, we rarely run some things and then start a program. Almost never happen. What do we do? We'll talk about it later. Alright. The other fork system call looks like this before. Can you get a process ID and return? But here's the difference in the same. The difference is the PID of the child is newly allocated in different that the parent. The child's parent PID is set to the PID of the parent process. Resource statistics are set to zero only in the child and any pigment pending signals are cleared and not inherited by the child. The child's signals doesn't inherit. The child does not inherit signals and they're cleared. But for the parent, all those signals are still there. Any required file locks are not inherited by the child's parents, still holds them. Yes. It's almost as if the process is sticking around. A common uses of fork is. To fork and exec. Why? Because we four can exact. You actually create a new process which then becomes the exact process. Extremely common. So you might wanna run something to clean something up or whatever, whatever you're trying to do. Right, You're working, you're working, you're working. Oh, I need to run this thing. So I'm gonna spawn a new process using fork. I'm gonna run. I wanna point out the not PID. Dave, This is why focus on last fall. Represents a fork in the road. If it returns PID that is not zero, that is in fact the child. If it returns zero, it's the parent. Wait, what? I'll show you. We do need to talk about a few more things. First, early assistance duplicated everything on the 4th, which is really time. We don't do that in real, real life. That is very uncommon. Lazy optimization strategy is called copyright. With the simple premise, if we're reading, no copy needs to be made. 4th child. Starts reading. We don't need to make copies of it. We can just use the same value, right? We have the code available to us. We can make these decisions pretty quickly. However, if he needs to, right? You're gonna need to actually modify it. You're gonna need to actually make it go. That's what we talk about. What do you mean well? When you for all the system processes you have. The child has access to all these things. Turn off. So we have to make, if we have a variable, global variables, the child and the parent have access to that global variable. Now if you're just reading it and you're not modifying the global variable, we can just keep the same what? We don't do anything with it. But if we need to modify it, when we do, the child gets their own copy of it, they're not actually modifying the same Variable. It's actually two processes, 2 variables and two different places. That's all I'm worth right terminating. We can type exit and we can give it a status. We'll wait. What? We get the status response? Other ways to terminate the classes way to terminate the program is simply let it finish. We call it falling off the end or implicit exit call. Good practice is either used exit or have a main return. Main return identifies exactly where your programs gonna finish. Right, so main returns return zero. Everything's good, right? Yay. You could also use exit if necessary as well. These are implicit or explicit ends to the program and why it's considered good practice if it's not obvious. Should always use exit, right? So obviously we're in here or in there. The process can also terminate. Process is set that signal SIGTERM or SIGKILL. We're gonna talk about signals later on and you can incur the wrath of the kernel. Many of you have had the wrath of the kernel imposed upon you. Segmentation fault as an example. Right. Exactly. Anytime your program terminates without like whoops, something wrong. Pointers on play, something goes wrong. That's inferring. Now bad news, Colonel. Doesn't tell you what went wrong. Colonel expects you to figure it out, yes. It has this error code that says segmentation fault, but that could be anything, yes? You can also register. This is also fun. You can also register function to execute upon termination is. Think of it as a clean up. Is your job developer? Some of you guys are. See developers and so forth. There are destroying methods and see C++. There is a Java terminate function that you can also implement that allows you to, upon exit, do something when an object goes out of. No, no, no, no, no. There really is. There's a there's a destroy function in Java. We just never use it. We just don't ever use it. But it is actually capable of using. You are capable of using it. You can specify. See, does not have automatic facility. There are no objects in. See. Nothing like that. But you can register function to execute on termination. If you have a clean up operation, you need to run function to execute in the reverse order with the register and these functions cannot call exit. The prototype for function is void function void and you can pass the name of that function to at exit and it will in fact run that. We don't do this unless we're using Damons, right? Like we don't do it that often, But with Damons as an example, you wanna run certain cleanup things. You're right, yes. Waiting for terminated children. When the process terminates this kernel, the kernel sends child to the parent. We'll learn about signals later, but that's not that important. When a child terminates before parents, it becomes a zombie process. Meaning nothing is waiting for it to finish. That's the more important part. You have to wait for it to finish, waiting for the parent to inquire about its status if the parent never inquires about the status. What happens? It stays as a zombie forever and ever. This process doesn't usually much resources, but it stings there. Once a parent acquires, that's when the child actually sees to exist. We have to wait. And we have to wait till otherwise that thing was forever. Wait and macros. We'll talk a little bit about this. We have PID, weight, and return to status Returns PID, the terminated child, or -1 error if not null. The status pointer contains normal additional information, so you can use these macros if. It says with exited with signal left. Forget those stands. That's usually wait. Yeah, I forgot you did meet the W week. But anyway, no, it's wait if exit if signal if stop. That's just an if statement, right? But these are macros. These are not real functions, but they will return a value. Did determine if it was a core dump. To determine if it was a stop, signal the terminal symbol. We use these, but not at all. These are really important for damons that are doing some sort of file operation. Those type of things you need to clean up. You need to make sure you exit appropriately. Right, so make sure you stop in a good state. You need to do it launching. This is another thing you could do launching and waiting. This is a four and weight all in one. Wait, what? System has a car command. Put the whole command in there Ch mod 755 right? Yes, it runs that program and then waits for it to finish, and then it's when it finishes. I'm not sure what that basin for. OK. But yeah, so system is a fork and a week. OK, usually your best best practices generally encourage the use of least privilege. What's the least privileges? You should only have what privileges you absolutely need at any given time. Why? If they compromise people, they're gonna compromise. Whenever you have access to. So if you run with less access, the compliment the once compromised will give away less results. Yes. In other words, if something goes wrong. Maybe it's not going to work, yes. So don't run everything. That's sometimes, though we do need higher privileges. We have to do something as we need to do this or we need to do that. So what do we do? Right now I want to say something else about this. Half the time you increase privileges to root. That's this your purpose of doing this? Anyway, the other half of the time you're actually sorry, you actually switch into a lower what's actually considered no privileges, OK? What do you mean? So is it Damon if Damon doesn't need any privileges other than to run? Really drop into nobody status and I access to nothing on the file system. Right, we can handle. You know network requests and then return responses, but we can't do anything, yes. Yeah, kind of. Yeah. You can increase the admin if you need to do something you need to do in admin. But the other half the time you're actually not running this the actual user. You're running as a even lower level user, right? Because when you log into the system, you have access to. The. Who directory? Typically, which involves being able to write to the district, can add files, you can manipulate files, and so forth. But if I give you a nobody user and I run as nobody, nobody has no privileges, they don't have a home directory, they don't have anything that they can write to the whole system, and that is safer, OK? So most of the time we either go down. Or we go up, but what do we do when we go up? And come right back in. You do the thing, come back down, you don't stay up there, huh? Humble yourself, Yeah, there you go. You know you need the higher privilege to do something, then do it, and then go back. Why? Because if you're if application is compromised or something goes wrong. You would rather have no privileges when something goes wrong than a bunch of privileges, yes. Alright, so real effective and save you your ID. There are four user ID's and four groups associated with these process, right? But it's something important we need to know. Cell. You have a real ID. This is the UID of the user who ran the process. Set to the parents real ID and never changes. Doesn't change. It doesn't need to change. We always need to know who started the process. That's. Effective UID. That's the UID that the process is currently wielding. This is the process that's checked against the operating system to see if they can do whatever it is that they're doing. The effective ID does what we use the permission checks based on. All the permission checks are based on. Initially the same as a real ID. By executing this set binary we'll talk about this. The process can change the effect of the effect it can change. More exactly is changed to the owner of the executable file. It's not change to random. There's actually a very sad things that we can do. And it requires system administration work to make it even even that. So. And then you have the saved ID. This is the processes original effective UID. Find exact call the effective UID sentence safety. Now why do we have that? And we also missing. D. No, we're good. Yeah. OK. It says 4, but. Yeah, this analysis. Anyways, whatever. The same ideas the original. The reason why we need this is on exact call, even if you've got additional privileges, we've got lesser privileges. The effective ID for that exact call is said to the same, right? Alright, changing real save, you can set UID. Set the effect of UID GID. GD is rooted in, saved in real or also set. Now root can use any value. If you are rude, you can change set UID to be any number. Yay, however. That's really practice, right? That's bad idea, yes. Most of the time you're some other user. And you're gonna set it to be Ruth briefly, yes. Non root must either use REAL or save UUID, there's no option there. Real or the effective and the real idea is the user of the process he ran it. And change it back to yourself. You go wait a second. What about affected by ID? You can change it to the owner of the executable file. So if you change the owner of the executable file to root. Then it can actually. Go to root if you're executing something else if. So we'll talk about this a little bit more detail when we actually have examples setting effective. Um, this word? Yeah, same idea except set E like. Yeah. No, I have this instead of set the effective, but this is actually changing the real estate, changing effective. This is what we're talking about a minute ago. It's just setting you like, alright, obtaining them, you can get them and you get the effect of here. Pretty simple. But now we're gonna get into a little bit more complex. Sessions and process groups. Each process is a member of a process group and one or more processes associated with another four job control. They're actually done this so that we can, if we have to terminate something, we can terminate all the things associated with. We have to kill a shell that's supposed. Hey, Cortana, what's? I'll say Karen logs in. If you wanna make Karens login go way by kill the idea, the process ID of her shell, every program she's been running, it doesn't matter what she's been doing, goes away, yes. Easy for system administration. That's kind of why we're you This and the things we do, however. The primary attribute of this sin signals to entire group at once. Each group has a word, We call it process group leader. A section is a collection of one or more processed groups. These are usually tied to the shell. You don't typically have a single process group. You'll have a bunch of existing consolidate logins around terminals, process groups. Divided one foreground process group and zero more background groups and each session has the session leader? Yes. The session is really the show. Yes. We kill the shell. To all process groups associated with that show. That's what we're trying to do. Why do we do this? System administration task makes it really easy. If you think somebody's doing something shouldn't do, you shut them down pretty darn quickly. Yes, very very, very quickly. Right back when I show you PSA, UX, blah blah blah. You run the command processes on the computer. You can very quickly see list of processes. There's a process that runs in the background. Normally it starts to boot and then our generally run. There's a special user, think root, patchy, and so patchy might be a web server. Usually runs as that user. Patching. Patching has right privileges, privileges only in certain areas. Was built to do. We handle system level test as convention named usually ends in the D and general runs this two requirements. When is the child of the night? Why? If we run as a net. Then if somebody kills a terminal doesn't kill us. We we started blue time, so we should remain running and so the system shuts down. When is this system shut down? When a niche shuts down. So we want to only be a child with a net. That way you kill any other process. You won't kill me. That's kind of one thing it's about. Must not be connected to a terminal. If it is connected to a terminal in any way, when you log out, the damn it goes away as well. This one just. Yes. Steps to become a Damon Recall 4. The parent calls exited some point cleans up the parent and chores the parent. Isn't it when the parent goes away for the child? The child inherits a nick as its process, right? And Nick automatically sits there and cleans up any processes that are finished. So does that by design? This way, right? You can. Wait, what do you mean? So if you have a parent that somehow dies, and you have a bunch of children that are doing work, when that work is done, they won't become zombie processes because. When the parent goes away, they become units children, and then it just sits there, leaves everything up. That makes sense. Then we call it set SSID. Demons don't want anything to do with process groups or sessions, so it's necessary to create your own. You're gonna actually create your own session ID. Change working directory for change directory. We won't wanna wish to lock anything by accident, so we typically change directory somewhere else. We close all the strippers. We don't want any inheritance relationship, all the scriptures when we do this. If we need to open something later, we open it later. Yes. We open file descriptor 01/2 and there usually redirect them to dev no or to a log file. Typically we do not do anything. Now you go wait a second. Two things. One the first two of these. Are to create a process and to make the parent process the right parent process. The first two right that's. The first one is hey, we're gonna we need to process. The second one is we need it to be the parent. The third one is to get out of the process groups we that disconnects us from the terminal that disconnects us from. Anything that might interfere with the closing the file scripts open file descriptors later we can do this. Simply to disconnect. From the terminal as well, because if you're open to if you're open the file descriptor standard inner open 01 and two are standard in standard out standard error 012 respectively. As a result, you're still kind of connected to the terminal. Now, could I convert the error to a log? Yes, close it, reopen it and send it to. Still should open it and close it. I still should close it and reopen it. Sorry. They still should I still should close all file descriptors and reopen it, right? Cause that's partially disconnected me from the terminal as well, because otherwise standard end which is 0 standard out which is one and standard RS2 are still connected to the terminal. So I'm gonna get rid of that and that's another way to do that. That's just the parts that. See. Most Unix systems we can implement most of this with. Doesn't do it all, but it does. The exit, excuse me, it does excuse me, it does the 4th exit, it does the. Set SSID, I don't know if it. No change directory forgot that's what that is. And then no clothes whether it closes Bob scripted or not. Maybe there's a file descriptor you kept open for some purpose, so you wanna keep that open. That is a way to set that. And it does all those things for you. And why is it do all those things for you? However, I've seen very few people actually use it. Why is that? I was always Debbie. I might do something slightly different than another system and. What happens if that difference makes a difference? I run into trouble. So what people do is they find out whether or not their system actually supports what they think it supports, and then they do those things. OK, alright, so we're gonna summarize. We've discussed process fundamentals, including process concepts, process components The one thing that we didn't really seem like it was really not interesting, and it really didn't seem very interesting, was this guy. OK. This is hard. We're gonna get into nondeterministic programs pretty quickly, yes, And you go. What do you mean? We're gonna actually be able to fork. We'll have two processes executed the same time, possibly doing two different, very different things, outputting separately, but not sure which one supposed to output first, yes. Yay, we'll show you an example of a program running that produces output. The interleaves with its parent. Exactly that interesting. It's valuable. It's important. I had a friend that really believed he didn't like to do clean up programs and he felt it was easier to, you know, issue a system command. So what he would do is he would exact. Or your system. Assuming the fork and then it would use system. To do whatever you wanted. I said what? That could be a problem for a couple reasons. Nothing went wrong. No. I mean, theoretically, if you really control the system and you really control the system, you can actually make that work and actually have no bad outcome. You really can. But what happens if somebody changes something in the system and then it spaghetti code? When you have dependencies like that, you run into situations in which. Tendencies may not work, but that's always true. Imagine I'm writing to and using the directory ETC. Slash. I keep saying your Mama, but that's funny. Anyways, BTC slash Mama right is an example, right? If from whatever reason somebody decides to make it, ETC, Your dad slash among them. Program fails, right? So there is a little bit of that happening. Go back Now if you have an external program that might actually be easier to fix that system call than it wouldn't be the what you call it, right? You can make a. Alias. These are all bad solutions, right? There's a little fragility here whenever you're opening certain files, are opening directories, and so forth. That's why we keep demons as dumb as doornails. And you don't want them doing massive things right. You want their configuration to be in a file. Right, not hard coded. So when it opens up it uses this configuration file. So if something changes you can control C kill the statement. Change the configuration file, restart it. It should come up with a new configuration. Yes, that's why we do that is because it it causes too many dependencies on the system, right? And we don't wanna we wanna generally avoid that. But this does not look revolutionary. This is actually probably the most important thing in the slide. Also, I always had fun with the add exit. I always thought this is the most fun thing in the world. You get to run programs like quick. But Damons are kind of what we're working towards, right? We wanna make a web server. We wanna make it this. We wanna make it that. We wanna make a system program that runs forever, right? We'll talk about clean up, we'll talk about handling tables, We'll talk about all those things associated with it. We can even do timing events, right? There's something happens every Tuesday 2:00 PM, right? Happily. I'm usually not even here on Tuesday at 2:00 PM. We talked about process termination concepts. We don't really get too much into it because we really don't really like to kill processes that much. We joke about it, but we really don't. If you're killing processes, that should be a system administration thing, not a. Can you kill things? Yes. We'll do that with threads more often. Do with processes. You certainly can kill processes, right? Kill people, right? We don't. We just don't do it that often. Usually the system administration might have left. For example Apache, Well it does create a daemon service that runs on its own away from the process group. It creates its own process group. So often want to kill all of this threads? We can do so for you. As an example. So while there's a lot of this stuff doesn't seem revolutionary, it is actually very very important that set UID is actually not trivial at all. You can do it all you want to and it will none of it will work. Right. You're like wait, father. It's because of the rules of which we can do set the user ID. OK, so it doesn't seem that interesting some of the stuff we did right changing and so forth, but it actually is. You'll also see a couple of commands that did not make it into the slideshow because slideshow. Made this many many many moons ago. But we'll be able to get user ID's. Will be able to accept that user ID. We're gonna be able to get the user ID. On the. Where you take the user ID and get the username. That's what I'm trying to say, yes. So that that will make our program looks a little cuter. Right. 1000 over and over or zero. Over and over. We'll actually be able to do that, but the demo presentation will be. Right but the point is is that it'll feel like we're giving like 4 demos right we'll give 4-4 different playing with users and groups set UID UID those separate things and then why would we