

Information Security-3
Prof. V Kamakoti
Department of Computer science and Engineering
Indian Institute of Technology Madras
Basics of Unix and Network Administration
Operating Systems Introduction
Mod01 Lecture 05
Module 5: Process and Threads

So welcome to module 5 and in this module we will be talking about process and threads. These are the next most visible part of the operating system that when you start executing a program you create a process just to give you a background of what we have covered in the previous modules. I have a program say hallo world dot c, I compile it, it gives you an A dot out. So hallo world dot c is a program, A dot out is an executable program. So we may call hallo world dot c as a source program or a source code and A dot out as an executable code. The moment you say dot slash A dot out and press enter then a process is created. So a process is nothing but a program in execution. Now there is a settle certain (0:55) distinction between process and the another unit of execution what we call as thread.

Now a process can be multi-threaded, in the sense that a process can have many units of execution. So what do we mean by units of execution? I will give you an example and one of the best example is that I could think of is the Microsoft word. The Microsoft word in its entirety (1:19) can be treated as process can be looked at as a process and when we look at the different functionalities that is done by Microsoft word, there is a functionality that accepts the input text. There is another functionality which formats it; there is another functionality which basically draws red line for your spelling mistakes. There is functionality which draws green line for your semantic or grammatical errors and then there is another functionality which basically stores your program.

So so at least we could think a five different functionalities or five different functional that are part of this process and is not necessary that all these five functions are done by one single sequential code and that gives us a notion of what we call as a multi-threaded program, where each one thread will be doing one of these functionalities. So when you invoke Microsoft word then there are five different execution units that are in action. One unit will be accepting the test, one will be formatting the test, one unit will be drawing the read lines, one will be drawing the green lines and another will be automatically saving it at regular intervals, each one of this can be called as a thread.

Now so in essence a thread is also a code in execution, right. A process is also a program in execution; thread is also a program in execution. So what is the difference between process and thread and this is extremely important for us to understand even from an information security point of view, because many times the hacks or the leakage of information can happen, because of improper shearing or not a very (0)(3:01) rigger rule base sharing of data between these different units of execution namely threads and processes. So before going to into the explanation of what is a difference between a process and thread. Let me also give you a background of what we have covered in this information security series so far.

So in the information security 2 course we have talked lot about computer organization. So many of you would have read two subjects in your curriculum, one is called computer architecture another is called computer organization, some university curriculums have this has computer architecture and the organization. We need to make a distinction what is computer architecture as a subject and what is computer organization as another subject.

Now the computer architecture basically it tells you how hardware is design, but computer organization basically tell you; how the compiler and the operating system can actually use hardware that is design. So the computer organization basically talks about the interface between your operating system compiler and the hardware, while the computer architecture basically talks about how you build that hardware to meet those functionalities.

So in the second course we have talk lot about computer organization, we have talked about memory management, we have talked about process management, we have also talked about interrupt management. So in the memory management, we have talked about segmentation, we have also talked about virtual memory or paging and we have also taught you how to write interrupt service routine for one of the most common architecture that is in practice the (0)(4:32) architecture, I strongly suggest that go back to the videos of the information security 2 course, which is available for free and look at those modules and if possible do those exercises if you do those then appreciation of what we have going to talk in this course will make lot more sense. These are available free in you tube, thanks to IIT Madras and the mooc platform and if you have problems in accessing that please put a note in the discussion forum and we will be too glad to send you the link, right.

So with this there is a background and with an assumption that you will make and earnest (0)(5:11) attempt to go back and look at those videos, I am now going to the next part of this

course namely process and threads, right. Now what is a process? A process is a program in execution. A thread this also a program in execution.

(Refer Slide Time: 5:27)

The slide is titled "Process vs Threads" in red text. It contains a list of similarities and differences between processes and threads. The similarities include sharing CPU, sequential execution, creating children, and non-blocking. The differences include lack of independence, shared address space, design for assistance, and separate user origins. The slide is numbered "2" in the top left corner and has "Operating Systems" written at the bottom.

- **Similarities**
 - Like processes threads share CPU and only one thread active (running) at a time.
 - Like processes, threads within a processes, threads within a processes execute sequentially.
 - Like processes, thread can create children.
 - And like process, if one thread is blocked, another thread can run.
- **Differences**
 - Unlike processes, threads are not independent of one another.
 - Unlike processes, all threads can access every address in the task .
 - Unlike processes, thread are design to assist one other.
 - As processes may originate from different users they may or may not assist one another.

Operating Systems

Now what is the like a process, which is actually a program which executes one instruction after another, the thread also will execute one instruction after another. A process can create children, right. So when you boot a Unix system then there is a what you call as the first process that comes up and that process actually creates many more child processes. So, the process is like a tree. There is one process that comes into execution when the system is booted up and there are many processes that are spawned. The spawned means created or fobbed. So different names are used in the operating system (())(6:07) parlance and these are the process that are the children child processes like, a process they thread can also create children and an like process if one thread is blocked another can also run, but in some seen arrows.

So a process is a program in execution, a thread is also a program in execution. So that is so whatever a process does during execution the thread can also do and that forms those four similarities that see in the slide, but what is the difference. So process is so, let me give the difference by an example, right you takes Microsoft power point and Microsoft word. They shear very less information or many times they do not shear any information. So these two can be treated as two different processes. They are two different executable programs and they execute and many times they execute in isolation, but when you take the other example of a Microsoft word in which, there are five different execution units running then they shear lot of information, for example, the thread that accepts input, the thread that formats the

input, the thread that drawn green lines for that input, the thread that draws red lines for the input and the thread that actually saves that input, note that in all these five sentences I have use that word input that essentially mean that this input is being sheared.

So a thread shears lot of information among itself and they collectively try and do a single job, while the processes can be two different entities doing two different totally different functionalities and may not and need not shear any information. This is at a functional level, but when you even when you look at an architectural level, threads can shear many resource, for example, the micro the five threads that we have talked about inside Microsoft word as an example, they can shear the memory, they can shear part of the stack, they can shear the code segment, they can shear the data segment some part of the segment data segment etcetera, but then they will have there some part which will be executing their own code and then they will have their own stack and each one can may not shear the general purpose registers, because when they execute, but there is a lot amount of data that five threads actually shear while, this is not the case when we look at two individual processes.

So the main difference between the process and thread from the architecture perspective and also from the functional perspective is they shear more threads shear more information while processes do not shear among themselves and the second most important thing that you should look at is that these processes is two different functionalities, two different processes to, two different functionalities, while threads actually work together to achieve one functionality. So this is something that we need to keep in mind. Since, the thread is a part of the process and it need not carry that much resources specifically allotted to it. A thread is called a light wait process.

(Refer Slide Time: 9:19)

But Why Threads?

- A process with multiple threads make a great server (for example printer server.)
- Because threads can share common data, they do not need to use inter process communication.
- Because of the very nature, threads can take advantage of multiprocessors.
- Threads only need a stack and storage for registers therefore, threads are easy to create.
- Threads use very little resources of an operating system in which they are working.
- Threads do not need new address space, global data, program code or operating system resources.
- Context switching are fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

Operating Systems

Now why do we need threads? So there are many many interesting things that we need to talk of here. The some of the things that I have already explained and some of the things that we will see here. Now a process can have multiple threads and these threads share common data, but then they can work in isolation one after another. The threading can happen at two levels, one is at the user level another is at a kernel level. Now let us this particular slide though, I am talking about many things that we have covered previous slide to my lecture, I will now make distinction between kernel level threading and user level threading and that will explain you why we need threads and what are the advantages and disadvantages of a having threads.

Now the operating system actually works in two modes, one is the kernel or supervisor mode another is the user mode. In the user mode, the operating system runs certain programs and if a process is executing in the user mode, it has very less privileges then when the process that is working in the kernel or the supervisor mode. So now when we want to move from the user space to the kernel space then what we do is a context switch in which, a user level program is basically removed and a kernel level program is basically scheduled.

Now the switch from user mode to a kernel mode or a user mode to a supervisor mode will takes lot more time. In the information security 2 course we have basically given lot of real examples and also, there are exercises which you can encode which basically teaches you how we can do a context switch from a user mode to a kernel mode. There we were talking about privilege levels an Intel or an AMD (11:21) its processor works in four privilege levels privilege 0, 1, 2, and 3 while the privilege level 0 or 0, 1 and 2 can be supervisor mode

while the 3 can be user mode. Now what it means to move from privilege level 3 to privilege level 0, 1, 2 we have covered that in great detail in the information security 2 course, I suggest that you go and look into it.

Now why are we talking about a kernel and you know user mode at this juncture (11:49), the point is when I want to implement threads, there are two ways by which I could implement threads. One is called user level threading, another is called kernel level threading. What is user level threading? The threads are created by a user program. So we have Pthreads etcetera and the way we can implement Pthreads is that we could create it at a user level we can also create threads at kernel level.

Now when I do it at a user level then when I want to context switch from one thread to another, so the threads do some multiple functionalities, when I want to do a context switch between the threads, I need not go to the kernel level, I can context switch at the user level itself. So the context switching actually becomes fast when working with threads at the user level. Why it is fast because when I do a context switch at a user level I do not have to save a lot of memory, lot of information that are required in with respect to memory need not be saved you may have to save your program counter, your stack pointer and some general purpose registers, but you need not save a lot of things about the address space.

So for me to move from one thread to another is going to be much simpler for me and much faster for me when I use thread at the user level threading, but when I am doing at that user level, one of the major disadvantage that you should keep in mind is that the kernel actually sees only one process, but the process could have multiple threads since, these threads are created at the user the kernel will not know that I have five threads, the kernel will only see one process.

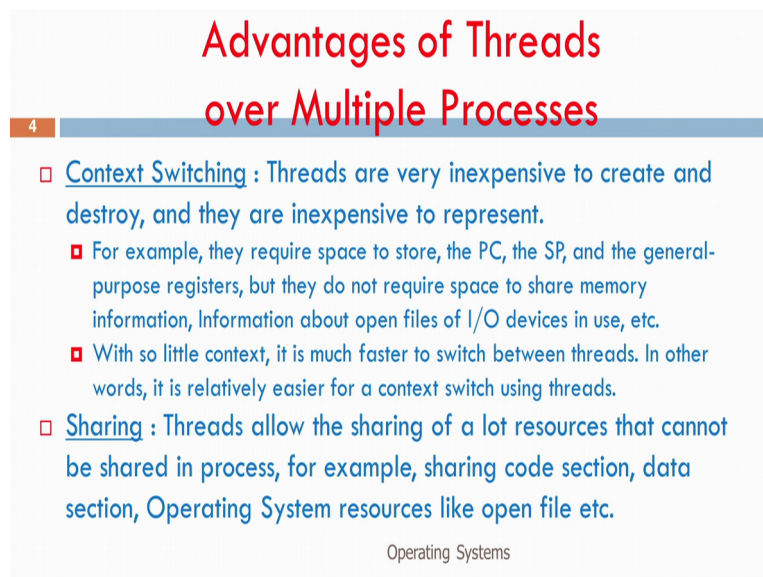
So even if one of my threads wants to execute a system call then basically it calls the kernel, the kernel will see that, that request is coming from the process, not from an end user side. So it will go and install or suspend the process for executing the system call. So what happens is if I have a user level threading and if one of my system one of my thread wants to execute a system call namely, it want to execute a system call, namely it want to write a disc or read from disc then all the other threads will be suspended by the kernel.

So the advantage of user level threading is context switching will be fast, but the disadvantage is if one of these threads has to do a system call then every other thread will be

installed by the operating system. The other part is the kernel level threading where the threads are created at the kernel level, when that is happening then if one of the threads wants to do a system call their kernel will know that this is the thread which needs the system call and it will suspend only that thread, the remaining threads can still execute, but when I want to do switch from one thread to another thread, so I have to basically go to the kernel level and then do the context switching.

So the context switching becomes slower when I do kernel level threading. So the slide why threads, this slides basically assumes that we trying to do user level threading but and many of these point become an advantage, when you are doing a user level threading especially the seventh point here, it says that context switching are fast when working threads. This is true when we do a user level threading, but this is not so true when we do a kernel level threading. Nevertheless, switching from one thread to another whether it is user level threading or kernel level threading each faster than switching from one process to another process.(15:30)

(Refer Slide Time: 15:33)



The slide is titled "Advantages of Threads over Multiple Processes" in red text. It features a blue header bar with the number "4" on the left. The content consists of two main bullet points, each with a blue square icon. The first bullet point is "Context Switching : Threads are very inexpensive to create and destroy, and they are inexpensive to represent." followed by two sub-bullets with red square icons: "For example, they require space to store, the PC, the SP, and the general-purpose registers, but they do not require space to share memory information, Information about open files or I/O devices in use, etc." and "With so little context, it is much faster to switch between threads. In other words, it is relatively easier for a context switch using threads." The second main bullet point is "Sharing : Threads allow the sharing of a lot resources that cannot be shared in process, for example, sharing code section, data section, Operating System resources like open file etc." At the bottom of the slide, the text "Operating Systems" is centered.

Advantages of Threads over Multiple Processes

- **Context Switching** : Threads are very inexpensive to create and destroy, and they are inexpensive to represent.
 - For example, they require space to store, the PC, the SP, and the general-purpose registers, but they do not require space to share memory information, Information about open files or I/O devices in use, etc.
 - With so little context, it is much faster to switch between threads. In other words, it is relatively easier for a context switch using threads.
- **Sharing** : Threads allow the sharing of a lot resources that cannot be shared in process, for example, sharing code section, data section, Operating System resources like open file etc.

Operating Systems

So let me just sum up what I have talked so far yes, threads, the biggest advantage of having threads over multiple processes, we can say that I have five different functionalities. Why do not I create five different processes? Why should I create five different threads? The first thing is the context switching is very very inexpensive , I have explained that in great detail and I also suggest again I repeat that go back to information security 2 course and see how really a context switching can happen in the context of a process, you will understand how difficult and expensive it is. Then the next thing is that shearing is also very very important,

threads can shear lot of resources and there is need to shear this resources as I had given an example of the Microsoft windows platform.

(Refer Slide Time: 16:16)

The slide is titled "Disadvantages of Threads over Multiple Processes" in red text. It has a small number "6" in a blue box on the left. The content consists of three bullet points:

- **Blocking:** If the kernel is single threaded, a system call of one thread will block the whole process and CPU may be idle during the blocking period.
- **Security** Since there is, an extensive sharing among threads there is a potential problem of security.
 - Possible that one thread over writes the stack of another thread (or damaged shared data) although it is very unlikely since threads are meant to cooperate on a single task.

Operating Systems

The disadvantage as I have told you, blocking is a disadvantage. If I am doing kernel level threading blocking is not going to be there. What we mean by blocking if one of the thread wants to execute a system call, other threads are also blocked and this will happen at the user level threading, but if I do a kernel level threading then this blocking will not be a issue, but again whatever advantage that we talked off in the previous slide, in terms of faster context switching we will get a little, but degraded (())(16:42) when we go to the kernel level threading, but why are we talking about threads and processes here?

There is a big implication of security, because now I have different entities trying to shear a resource when an entity is sheared there is an extensive shearing actually not a simple shearing then there is a potential problem of security we will address this as we go through not just information security 3, but 4 and 5 and then 5 courses that we have planned in the series, lot of points will come out where the security is basically compromise, because two processes or two units of execution namely threads are trying to shear some information in which, one of the thread can be a malicious one or can be a weak one, which can leak information. So this is where security gets compromise.

So if you want to actually become a good security engineer, you have to be very thoro with this fundamental things like, what is thread? What is process? What is this involving you need to have thoro knowledge of that and that is what this course is aiming at and so please

have undivided attention in the rest of this module and understand this and again I repeat, I will be repeating it many times, please go through information security 2 course at the least if you do information security 1 and 2, the videos if you look and come to this course is be extremely proof full but at least to you need to look at so that you get the full effective delivery of this course from a understanding perspective, thank you.

(Refer Slide Time: 18:25)

7

What kind of Application Benefit from threading?

- A proxy server satisfying the requests for a number of computers on a LAN would be benefited by a multi-threaded process.
- In general, any program that has to do more than one task at a time could benefit from multitasking.
 - For example, a program that reads input, process it, and outputs could have three threads, one for each task.
- Any sequential process that cannot be divided into parallel task will not benefit from thread.
- They would block until the previous one completes.
 - For example, a program that displays the time of the day would not benefit from multiple threads.

Operating Systems

Next we will go on to what kind of benefit and application get about threading. So we will just see some examples of this. There is a proxy server is a very very interesting example of a multi-threaded process. So in general if there are programs that I has to do more than one task at a time that will certainly benefit from multi-tasking and so, the example again I have coated here is the Microsoft word type of program, but if there are sequential processes that cannot be divided into parallel task, right then these are the things that will not benefit out of threading, right.

So one of the interesting thing is that as an when the processor, the architecture, the hardware started supporting multi-threading. Now there is owners or responsibility of everyone starting from a teacher, who teaches a B. Tech course, a student who learns under graduate follow, post graduate follow then he becomes he or she becomes a teacher. So there is a generation responsibility that we have to guide the next generation in looking at concurrency looking at parallelism how to see, if there are components of your problem that could be solved concurrently and that is very very important.

Unfortunately, today many curriculums do not address this issue, hopefully over a period of time curriculums do improve and start looking at concurrent programming as a paradigm even at an introductory stage, even at your first year of your undergraduate program or your postgraduate program, concurrency should be taught how to look at things as you know different units of execution that could execute in parallel and if unless you do that you will not use the full potential of modern machines, today you cannot get a single core say, (20:07) its machine you cannot get the single core machine on your desktop, even your mobile phone we will go up to 8th processors. So what will you do with these things, right? We just having processors cannot be you know a doll, right. We cannot just keep playing you have to use it; it cannot be just a showcase doll.

So it is very very important that all of us learn concurrent programming, if they are teachers who are listening to me now, if there are decision makers university, board of academic courses people, who are looking at it, please please please go and start looking at introducing concurrent programming in your curriculum and that is very very very important and if there are people who are not looking from this and you people who can make decisions please, pass this video to them.

(Refer Slide Time: 20:59)

Action of Kernel During Context Switch

8

- [Context Switch Among Processes](#)
- Before a process can be switched its process control block (PCB) must be saved by the operating system.
- The PCB consists of the following information:
 - The process state.
 - The program counter, PC.
 - The values of the different registers.
 - The CPU scheduling information for the process.
 - Memory management information regarding the process.
 - Possible accounting information for this process.
 - I/O status information of the process.

When the PCB of the currently executing process is saved the operating system loads the PCB of the next process that has to be run on CPU. This is a heavy task and it takes a lot of time.

Operating Systems

Now I will summarize what do you mean by a context switch again, we have covered this in information security 2 course. So there something call a process control block a PCB. Now it basically saves information about a process and what is that information that you need to save? We need to save that information so that at any point of time, I can restart that process exactly at the point where it left and when I am restarting, the entire state of the system what

you mean by a state the program counter, the general purpose registers, the memory, the stack pointer etcetera should be restored at exactly the point I left.

In addition, so we have to store the process state. What we mean by process state the program counter, the PC and the process state is also whether when at the time of context switch what sort of why did I context switch we will see as we go into the scheduling in the next couple of modules, I context switch just, because I made a mistake as a process, right. So that means there is I landed a pin trap (())(22:01) I say I had executed divide by zero or I did a segment overflow or I wanted to actually take some help from the operating system, I want to print on the screen printf or scanf, I want to read from the keyboard, I want to write or read from a file. So I did a system call.

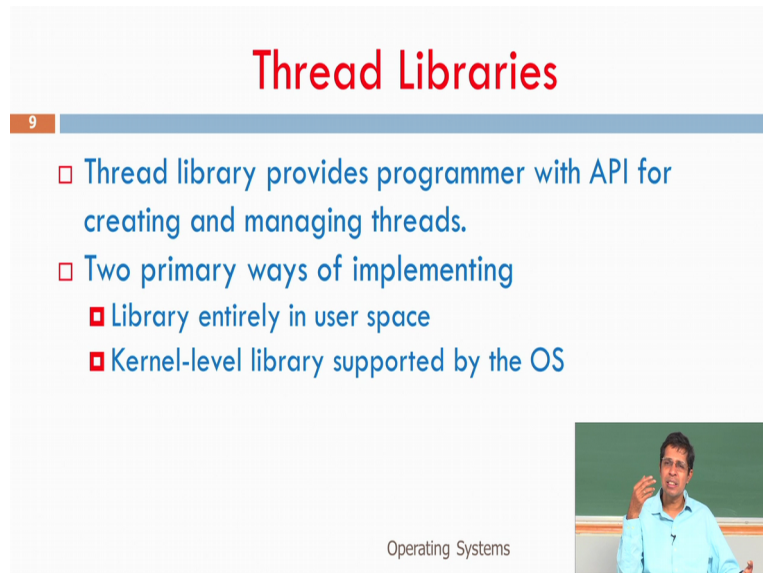
So in these type of things, I go into a suspended state and the operating system will finish the interrupt service or the system call and then get back. So my state would be a suspended state or a waiting state after which, I will go to ready state and I will come back. So a process can be in different state we will talk about those states very shortly. So we have to store the process state then we have to store the program counter, I hope all of you know what program counters mean then the values of the different registers, then the CPU scheduling information, then the memory management information like the page table etcetera and the possible accounting information for this process. How much CPU time I have took use at etcetera and then IO status information.

So all these things constitute what we mean comprise what you mean as a process control block and this needs to be saved so that the process can be restarted at any point from exactly the point where they left. So when we have multiple processes that are running then each process will have a process control block and these process control blocks will be maintain by the operating system and as an when a process one process one switches to process two, the context of process one is saved in its process control block and the context of process two is loaded from its process control block and process two executes.

Now process two finishes and again or process two is pulled out and process one as to now execute then again process 2s context is stored in process its process control block. Process 1s context is loaded back from its process control block into the system and it is starts executing. So the process control blocks basically maintain the context of all the processes that are currently executing if you are having in understanding what I have told in this slide, please

again go back to information security 2 course and there are we have explain everything in great detail.

(Refer Slide Time: 24:21)



The slide is titled "Thread Libraries" in red text. It features a blue header bar with the number "9" on the left. The main content is a list of three bullet points in blue text, each preceded by a square icon. The first bullet point is "Thread library provides programmer with API for creating and managing threads." The second bullet point is "Two primary ways of implementing", followed by two sub-bullet points: "Library entirely in user space" and "Kernel-level library supported by the OS". In the bottom right corner, there is a small video inset showing a man in a light blue shirt speaking. Below the video inset, the text "Operating Systems" is visible.

- Thread library provides programmer with API for creating and managing threads.
- Two primary ways of implementing
 - Library entirely in user space
 - Kernel-level library supported by the OS

Now like processes we could have thread libraries and I later let us now talk about there are two primary ways as you see, a library entirely in the user space and library at the kernel level which is supported by user. Now what is a thread library like math library, where you have lot of math functions already implemented like cos, hyperbolic I can do in C right. So similarly a threaded library, also we will have several things which can basically talk invoke lot of thread related functionalities, we will see a very simple program down the line, which will talk about different functions in this thread library.

Nevertheless when I want to use a math function, I say hash include math dot h and then I can use all the math function. Similarly, I can use hash include thread dot h and I can use several functions, which are related to creation, termination and spawning of threads, okay.


(Refer Slide Time: 25:12)

PThreads

10

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Operating Systems



Thread Cancellation

11

- Terminating a thread before it has finished
- Two general approaches:
 - Asynchronous cancellation terminates the target thread immediately
 - Deferred cancellation allows the target thread to periodically check if it should be cancelled
- Problems when resources have been allocated to a canceled thread or while in the midst of updating data sharing with other threads

Operating Systems

So this is the one of the very very important thread libraries the P threads , this can be supported at the user level or kernel level and it there is standard call the POSIX standard from IEEE, which basically gives you the application programmer interface for threading that creation and synchronization. So the what is application program interface, which actually specifies the behavior of the thread library and this is common in Unix operating system Solaris, Linux Mac OS X, you can basically use Pthreads there like how I could create a process and I could kill a process or I can terminate a process, I can also do a thread cancellation, I can terminate a thread and the thread can normally it can finish by itself, but it can also be killed we could have asynchronous cancellation, which means it will terminate the target thread immediately you could have a differed cancellation allows the target thread

to periodically check if it should be cancelled, right. The thread can itself go and check for itself.

So what is the main thing here? What is a role of operating system when a thread is cancelled then it should immediately, the resources that are allocated at the thread should be brought back, but then there are multiple threads that are sharing that information, see if a process is terminated then all the information related to the process can be released, but in the case of thread it cannot be if I am having a user level threading and one thread terminates then there are or even if I have a kernel level threading, one thread terminates. What is it that I could get out of the threads that is very very important. What is it that I need to release and if I fail to release then it becomes a leak, right because then it becomes nobody's resource.

So when a thread is saying, it is completed or it is canceled for then we need to retrieve the resources release the resources exactly into it and that is going to be a problem, because I really do not know, which are information that is used exclusively by the thread and which are the information that are shared. So that is also one very important implementation within the operating system, because if I am not going to release it properly then potentially I could have a leak of the resource and I could also have certain security vulnerabilities there. So thread cancellation is very very very important, right. So to create a thread is like making friendship, but to cancel a thread is something like to maintain it. So it is very difficult to maintain friendship and similarly, a thread cancellation also is very difficult much much difficult than creating the thread.

(Refer Slide Time: 27:52)



Thread Pools

12

- Create a number of threads in a pool where they await work
- Advantages:
 - ▣ Usually slightly faster to service a request with an existing thread than create a new thread
 - ▣ Allows the number of threads in the application(s) to be bound to the size of the pool
- Ex:
 - ▣ QueueUserWorkItem(LPCTSTR StartRoutine, PVOID Param, ULONG Flags) in Win32 API
 - ▣ java.util.concurrent package in Java 1.5

Operating Systems

One of the very interesting things that I have come up is thread pools where these are lot of threads are already created and they are all awaiting work, for example a printers pool etcetera, right. So since I created the work already. So there are some examples that have been given at the end win32 API and java 1.5, where you can basically use these thread pools. So since these threads are created already the creation is faster. So if we want to have a request, for example I have a pool of threads which will do certain functionality and I want to do that functionality, it is very quickly achieved by using one of the thread that is already created, just I have to give some inputs or a signal and it will work and it will basically give back the result.

So thread pools become extremely important in the context of the real time operating systems where you know we have looking at very quick response time, we have talked about what are real time operating systems and their distinction in the previous modules and one of the things that we said that we are looking at response time, we need very quick response time and these thread pools can be used for getting very good response time.


(Refer Slide Time: 28:57)

Thread-Specific Data

13

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

Operating Systems



And we could have thread specific data and for every thread each thread can have its own copy of data and this is done, because then this useful when you do not have control over the thread creation process. So when I have a thread pool ((29:10) so, somebody request for the thread, they take the thread they execute it and get back. So I do not control that as an operating system we have less control over that. Now the moment I have thread specific data, now it is very very important that when the thread completes, the entire data needs to get erased, entire threads may have to go off otherwise, there could be security vulnerability. So though threads are very interesting for us, threads need to be handled very very carefully in the context of information security.


(Refer Slide Time: 29:40)

Pthread Example

14

```
□ #include <pthread.h>
...
int main(int argc, char *argv[])
{
    pthread_t tid;
    pthread_attr_t attr;
    ...
    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, argv[1]);
    pthread_join(tid, NULL);
    ...
}
void *runner(void *param)
{
    ...
    pthread_exit(0);
}
```

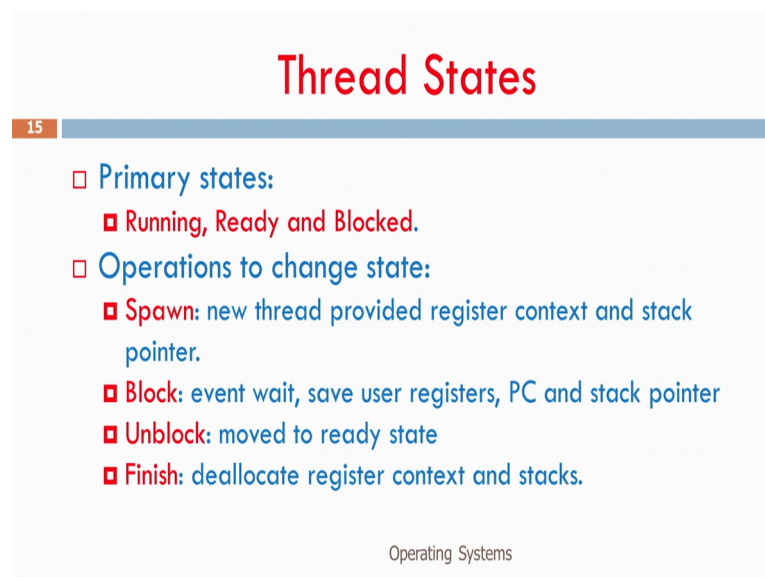
Operating Systems



Now this is the simple example of a P thread and what we see there is that. So 2 things are very very important, first thing is there is a P thread dot h as I told you math dot h we also have P thread dot h then we can create thread by using P thread t you can get a task id for this. Now the two functions, we will not go into the entire Pthread creation (30:03). Let us see look at two functions there, Pthread create and Pthread join. So basically the create will create task id and there you also have, so when I create a thread that means I say, this is say thread. In this thread executes this program.

Now what is a program that I am going to execute there? The program is call runner. So I am just looking at the 4th thing Pthread Crete tid and you please, note that there is a runner there. That runner essentially means another program you see some program there down for runner. So I can create a thread and say in that thread this particular code will be executed that is possible and then I can go and exit from a thread, you see Pthread exit inside runner essentially it will exit. So this is very simple example, I am not going into full details of what Pthreads actually do, but this is an example where we say that threads could be created and the threads could be completed.

(Refer Slide Time: 31:07)



Thread States

15

- **Primary states:**
 - ▣ **Running, Ready and Blocked.**
- **Operations to change state:**
 - ▣ **Spawn:** new thread provided register context and stack pointer.
 - ▣ **Block:** event wait, save user registers, PC and stack pointer
 - ▣ **Unblock:** moved to ready state
 - ▣ **Finish:** deallocate register context and stacks.

Operating Systems

We have talked about process state in some of the previous slide. So whatever process or thread they have state (31:12). So a thread can be running as currently execute in or thread can be waiting to getting executed it. So that is called a ready a state and the thread could be in a blocked state, where it is asking for some system service or it is asking which is doing an IO. So it will be waiting, it will be blocked till that IO is completed, it will be blocked after the IO finishes, it will brought to the ready state and then it will be scheduled for running.

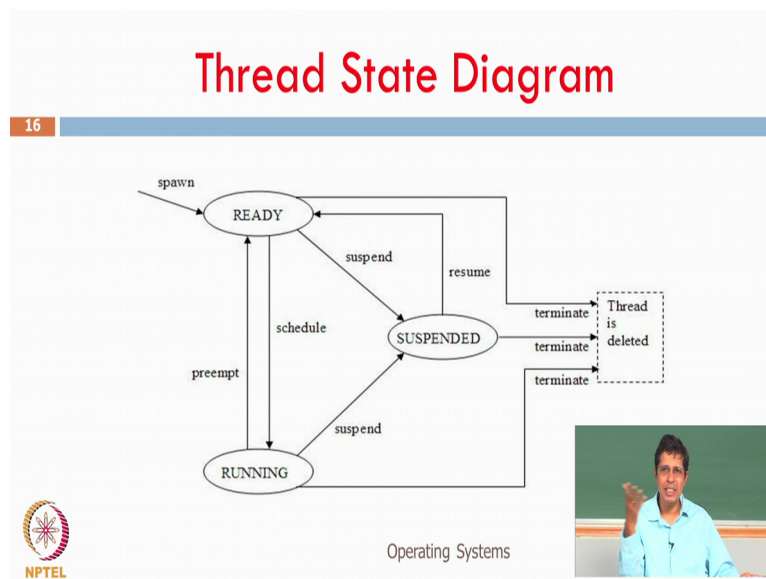
So running, ready and blocked are three states of a thread we will cover that in using a state state diagram in the next slide. Now so what happens is first a thread when you create you call spawn. Spawn means you create a thread. Now immediately when you spawn then it will go into a queue call ready queue. The queue will basically have all the fellows who can execute.

Now a scheduler if you are it is a user level threading then the user level scheduler will run. If is a kernel level threading and the kernel level scheduler will run and the scheduler will now of all the threads that are waiting, it will find out whoms would I execute, it will take it and put it into the CPU then it will start executing that time the thread is in the running state. Currently it was in the ready state then it becomes the a running state. When it is running, it say is it wants to do a printf. So immediately it calls operating system.

So now the operating system starts executing at this point it will take that state that thread and put it into a blocked state and it will do that scanf for that true thing and it will give the input of got out of scanf to that thread till know the thread is in block state, once that system call corresponding to the thread is completed then it is now again moved back to the ready state, again it goes to the running state. So this goes on this merigo round goes one after another till the thread is completed and then block and unlock or basically things blocks is to actually block that thread from running so that, because it needs some service.

So what is block do? It actually does a context switch from the thread to a kernel level routine. So in the case of a block then you actually save all the user registers, program counter and stack pointer, you save the context of the thread and move to the context of the service routine and once the service routine work finishes, you unblock that process and throw it back to the running ready state and when all your things are finished then you can come out all this.

(Refer Slide Time: 33:46)



So this is a thread state diagram when I spawned, I go to the ready state and when I am scheduled I go to the running state and for some reason due to an interrupt if I am preempted that is stopped, I do not do any mistake, I do not want any service, but then external interrupt has come and it say stop that is called preempt then again I go back to the ready queue and again the scheduler actually schedules me back, I come back to running again I go back if there is a preempt back to the ready queue.

So that are the two edges between ready and running state but when I am in a running state and I ask for some service system called I essentially get suspended I go into the suspended state or what you call as waiting state and when the call is serviced by the operating system and it is slower than I can go back to the resume state, even when I am in a ready state I told about user level threading all my threads can be suspended if one of my threads want so one of my thread is running the remaining 4 threads.

So let us say 5 threads are a part of a process. So 4 threads are waiting, they are in the ready queue one thread is running and this is thread now says, I want to have a system call then the remaining 4 threads also should be suspended, because the operating system only knows one (())(35:03). So this basically goes back and then we go to the suspended state and again when that system call is over I is resume and that any state I terminate if I if there is a need for termination it is user terminates or for some other reason gets terminate the thread is actually deleted.

So this is the thread state diagram. This is also be the process state diagram in some sense (()) (35:29) we will look at process state diagrams, when we look at the scheduling in the next day. So with we end this module, I hope you all understood more about threads and processes and their differences between them, the intention of this course is not to teach what processes are that is done in the information security 2 course. Now we just give you a different between processes and threads, right. So we will move on to the next module. Thank you.