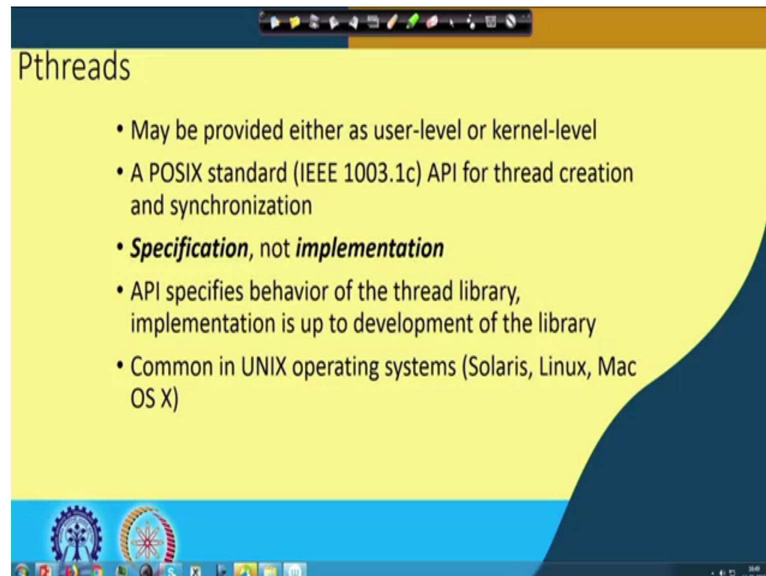
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 22
Threads (Contd.)

(Refer Slide Time: 00:28)



So, Pthreads is one of the libraries that is available in many of the operating systems that helps us in writing thread level programs. So, that can utilize this thread concept. And you see that this is said to be a standard and so, different operating systems may have different implementations of them. So, as a waste designer once concerned will be how to implement it, but as a user of the thread level libraries.

So, we have to know what are the specific calls what are the specific services that are provided by the thread level library for using them. Now so, naturally it is not possible to go into a full fledged discussion on the Pthread library. So, we will try to see the essential parts of it and try to get a feeling like how these thread level programs are developed and how are they; how are they useful for having some job done.

So, this is a POSIX standard. So, I triple E 1003.1 c and that is API for thread creation and synchronization. So, it will create threads and do synchronization between them. And as I already said this is an specification not an implementation. So, implementation

may vary, but specification will remain. This API it specifies behavior of the thread library implementation is up to the development of the library.

So, this specification will tell how what should be the interface library over the three interface library what would be the calls in the library, what are the parameters that this calls should have. And common UNIX operating systems like solar use Linux, Mac OS X. So, all of them support this Pthread.

(Refer Slide Time: 02:13)

Pthreads Example

- Next two slides show a multithreaded C program that calculates the summation of a non-negative integer in a separate thread.
- In a Pthreads program, separate threads begin execution in a specified function. In the program, this is the runner() function.
- When this program starts, a single thread of control begins in main(). After some initialization, main() creates a second thread that begins control in the runner() function. Both threads share the global data sum.

So, that is why it is one of the very popular ones that we have. So, next we are going to see a multithreaded C program that calculates the summation of non negative integer in a separate thread.

So, it will be from the main thread main process or main threads. So, will create another thread that will calculate the sum of non negative integers. So, in a Pthreads program to separate threads begin execution in a specific function. So, if you try to correlate like with the fork system call that we have in case of the fork system call that we have in case of Linux in case of processes. So, here we have got these threads to be in case of fork what was happening is that the new process that is created.

So, it has got this code data and stack segments and we said that if this is the code segment then if at the time of fork. So, at this line I have given a call to fork then after

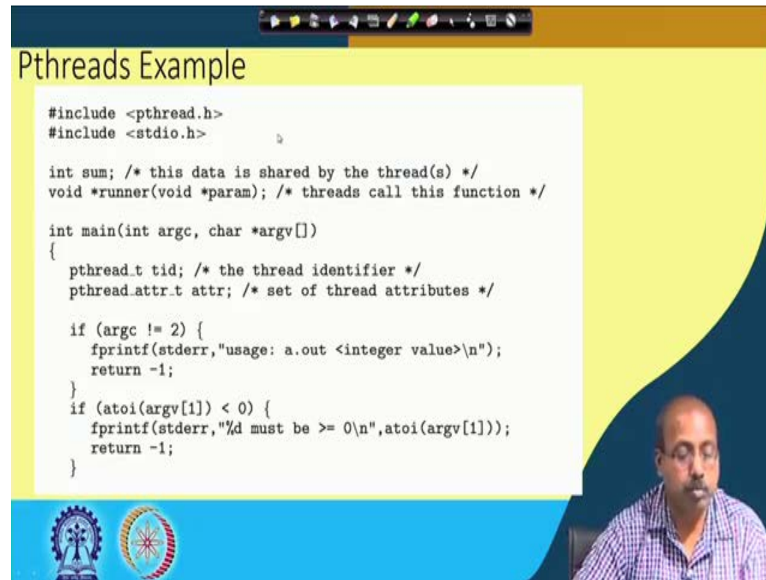
returning from the fork, the both the parent and child they will start executing from this point.

So, depending upon the flow of control this parent and child they may follow to different paths, but both of them will start from this point only, so, that is there. So, both of them will start from this point and continue like this. Now in case of thread what happens is that it is not like that. So, whenever you are trying to create a new thread. So, you have to tell which function it will execute. So, there maybe it will not execute the entire code so, we will tell a particular function to be executed by the thread.

So, we will see in this particular example that the thread that will be created, it will be a calling a function called runner which is a function within which is a function within the code and this thread will execute only this runner function. So, once the runner function is over the thread will also die. So, that is the thing. So, this that is how this within the same program we can have different different procedures or different different functions and each thread can be associated with one such function to start with and once that function is over that thread also dies.

So, going back to the example that we had previously like one application we can have this the networking to network to fetch the data, then we have got this we have update display we have got this spell checker and all that each of them can be different functions. So, we can create different threads which will be talking to which will be invoking those functions and once those functions are over then the thread is also over. So, in the program starts a single thread of control begins in the main function and after some initialization main will create another thread that will begin call the that begins control in the runner function. So, will look into the code that will make us more comfortable to understand it.

(Refer Slide Time: 05:14)

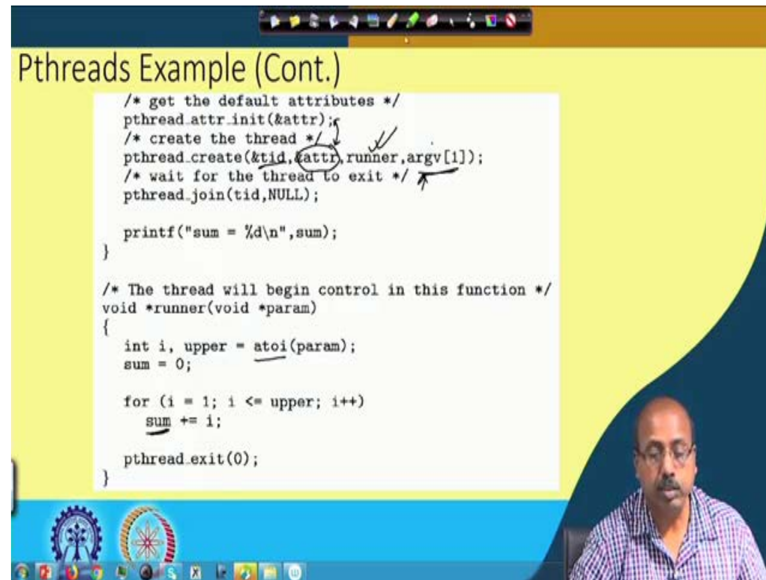


So, here you see in this program. So, this is the first thing that any thread pthread programs. So, it should have this hash include line pthread dot h. So, this is a header file that has got the definitions associated with this pthread types and all for example, this pthread t. So, this is the tid is a variable. So, that is the thread identifier. So, pthread t is the type of this is id type then attribute is the pthread attribute t is. So, this is the attributes that we that a thread will have so, that type.

Now we have got this sum as a variable and will have a function description of function definition here, which is a runner white and this one parameter will be passed which is of type void. Now in this main program what we are doing? So, if argc not equal to 2. So, we are giving in giving some error because he assume that the program will be called with some integer value. So, this value should be coming and then this will be call it will call this into it will be the function this program should be called with some integer value.

Then that the integer value should be positive the second check this atoi argv less than 0. So, that is checking whether the value given is less than 0 or not if it is less than 0 that is an error. So, we are assuming that this program will be called by giving a giving some positive value to it. So, either 0 or some positive value to it cannot be the value given cannot be negative. So, this is the format at which this program should be invoked a dot out and then some integer value.

(Refer Slide Time: 07:02)



Now, after this; so, what is doing what was then I am it is I it is doing some default attribute. So, it is getting some default attribute for this pthread. So, that is at a obtained in this attribute structure pthread attribute in it. So, this gets all that all this default attributes in this and then in this it is now it is going to create a thread. So, this pthread create tid attribute runner and argv 1. So, what is happening is that it is calling this function runner with this particular parameters. So, this thread will be created the format of this pthread create is like this.

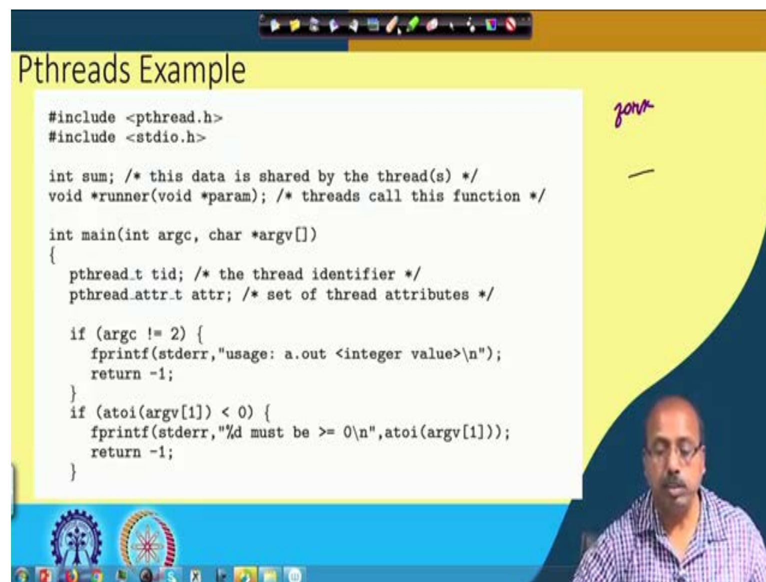
So, they I have to tell the we have to give this thread id. So, thread will be created like when this thread is when this pthread create is executed, system will give it a thread id and that thread id value will be returned to this tid. Then whatever attributes you want to set so, that has to be passed to this attribute structure and since we do not want to make modification to whatever is available by default. So, this is basically the tricks. So, we get take that the default values in this attribute variable previously and that variable itself is passed.

So, that this thread is created with default attributes. Then in the runner, so, this is the function that will be executed by this thread. So, that function name is given and then this argv 1. So, whatever was the value like say 10 20 30 any positive integer greater or equal 0. So, that value was given. So, that is the parameter that is passed to this function.

Now if we look into. So, what will happen is that this thread will be created and this thread will be executing this function runner with this particular argument.

And in this function runner what we are doing? So, we are since this is a text that is coming. So, there is a string variable. So, that is first converted to some integer by giving a call to the function atoi. So, this is in the variable upper I am getting the corresponding integer value and then I am just adding this numbers 1 to upper into the variable some.

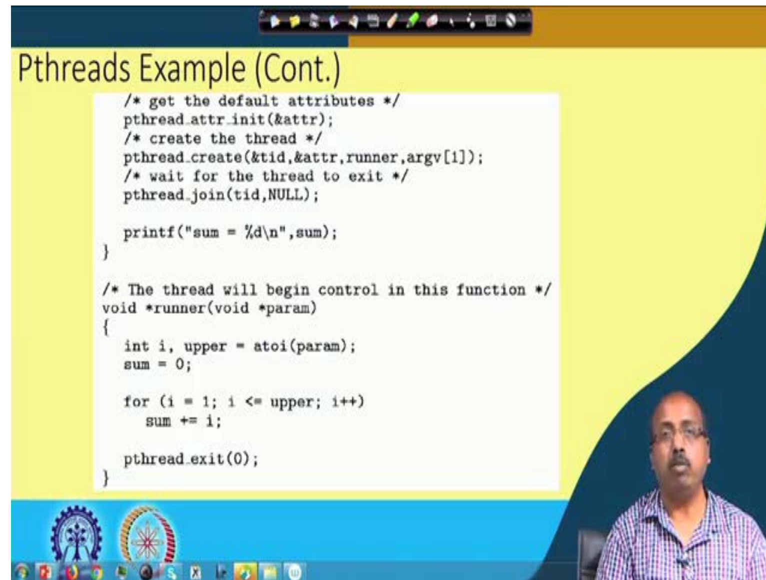
(Refer Slide Time: 09:35)



And if you look into the sum variable so, it is if you look in to the sum. So, this sum was a global variable. So, that was visible to this main program that is executing. So, as well as this sum is also visible to the. So, this is this was the main thread at which execution started and this main thread created another thread for running the function runner.

But both the threads can see the sum. So, unlike if process the parent child process that is there. So, if the a child process was created instead of creating a thread and that child process was assigned the job of computing that some, then what this child process will do. So, this child process what this child process computes in the sum so, that will not be visible to the parent process.

(Refer Slide Time: 10:25)



But since thread this data segment is common between all the thread. So, both of them see the same sum variables. So, as a result the sum that is modified here is also available to the parent the parent thread which created this one. So, we have got this thing. So, that is the advantage of thread programming. So, you do not have to put some on a shared memory location so, that both of them can access it. So, they can be the part of the data segment. Now after creating this thread the the parent thread does is that it is waiting for the child thread to be over. So, it will wait for the thread to exit. So, pthread join and this thread where though thread which was executing the runner at the end of its.

So, it executes pthread exit. So, this pthread exit. So, this will send a message to this parent thread telling that when it was waiting for this where it was waiting for this join. So, that this waiting is over and then this main thread it can print the sum values. So, this sum that is printed here will be the value that is calculated here because the variable can be accessed by both the threads they are part of the same data segment.

So, this gives us a very simple idea like how can I create a number of threads. So, basically the important calls that we have learnt is this pthread attribute in it, that initializes the attributes then we have got this pthread create for creating the threads we have got pthread join for waiting for the threads to be over and we have got pthread exit by which you can exit from a thread.

(Refer Slide Time: 12:21)

Pthreads Code for Joining Ten Threads

- The summation program in the previous slides creates a single thread.
- With multicore systems, writing programs containing several threads is common.
- Example a Pthreads program, for joining the threads:

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
  pthread_join(workers[i], NULL);
```

Now, for more details, so, you can consult this pthread library the manuals and all. So, this is another example where we have got a pthread code for joining 10 threads may be we have created; we have created an array of 10 threads and they are called the say they are put into this they are name the workers number of threads. So, this pthread join workers i dot NULL. So, all the process if the thread the parent threads that we have the main thread that we have. So, it is waiting for joining all of for joining of all this worker threads.

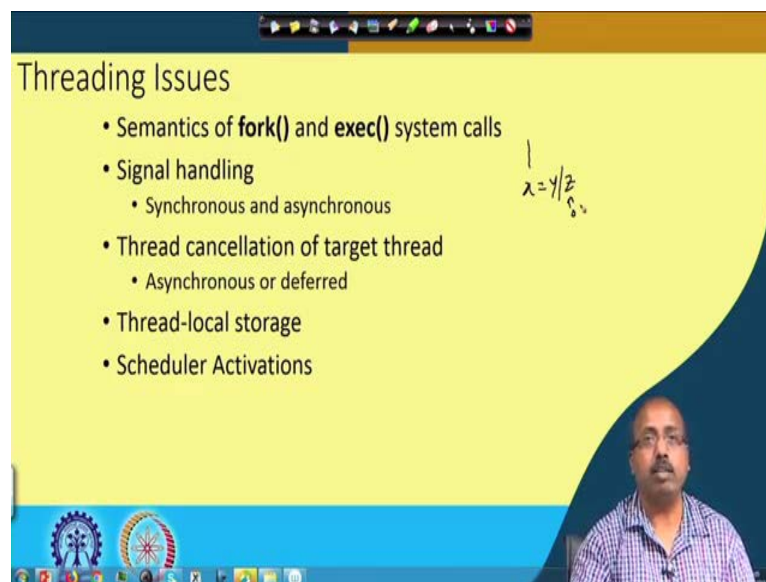
So, this summation program that we have previously. So, in the that creates a single thread in a multicore system. So, we can have several threads like each core may be executing a thread and some point of time we may need to do a synchronization. So, maybe we have got a number of threads which are going on parallelly on to different codes. So, this is C 1 C 2 C 3 C 4 etcetera then we need a point at which all this threads are over.

So, this synchronization if you want to do, then the main thread it should execute a statement like this ok. So, pthread join on a loop. So, it is for all the threads it will wait and once it is all the threads have joint then only the main thread will proceed. So, you can think of the situation like this for that matrix example that we look into maybe a one is doing that transpose. So, this is doing the transport operation the. So, this is doing that inversion this is finding the determinant etcetera.

So, now what can happen is that you can say C 4 is doing some say incrementing each element by some value, so, some value v. So, it is incrementing now it may so, happen that I have got a continuous flow of mattresses and once one matrix operation is over then again the operation should do should go on a second matrix.

So, once as I need to know that there is a main thread which you which creates all these threads and gives the job to them and once the threads are over threads done with the current data set in the next matrix will be read and again they will be there should be informed that now you should continue. So, somehow we need to we need to create the threads and join wait for the threads to be over. So, this has to be done. So, we can do it using this particular approach by this joining of threads.

(Refer Slide Time: 14:56)



Now, threading issues like we have to look into the semantics of fork and exec system call like, if what will happen if threading thread execute say fork call or an exec call then because thread is a part of process. So, there are multiple threads within a process. Now if one of the thread forks then the what happens to the remaining threads or one of the thread over lies over call makes an exec system call so, that the content of the process is overlaid then what happens to the remaining threads. Then there are issues are signal handling. So, if some signal is a situation where some abnormal situation has occurred and that is in informed. For example, if a program is executing and at some point of time it has got a statement like x equal to y by z.

Now, if this value of z becomes equal to 0, then that is a division by 0 error. So, this is an exceptional situation. So, what the system does is that the system sends a signal to the process that this process has got a division by 0 error. So, the process is expected to handle this signals carefully if not, then the process is simply terminated and this thing happen. So, what will happen if a thread gets a signal. So, is it visible to all the threads or it is visible to only that thread so, that is important.

Then how do you cancel thread or if you one particular thread we want to cancel. So, what happens to the others? Then the local storage and scheduler activation. So, these are various issues that we need to see when we are talking about these threads.

(Refer Slide Time: 16:51)

Semantics of fork() and exec()

- Does fork() duplicate only the calling thread or all threads?
  - Some UNIX systems have two versions of fork
- exec() usually works as normal – replace the running process including all threads

So, in our successive discussion. So, will be looking into these issues the first thing is that the fork and exec system call. So, does fork duplicate only the calling thread or all threads. So, this is the first issue that we have. So, have got several threads now, so, we have got several threads now this threads makes a fork call.

Now what will happen? Weather the new process that is created so, will it be consisting of copy of this thread only or it will also have copies of all these threads now this is of course. So, no direct answer can be given. So, it is it depends on the OS designer. So, some unique system so, they have got two versions of fork. So, there is a fork call and there is another call which is called vfork. So, this fork, so, this creates a copy of the process.

So, all the threads will be duplicated whereas, this vfork this is actually duplicate only the thread. So, you have got this thing similarly the exec. So, it usually works as normal replace the running process including all threads. Because the here if you replace one thread that is not possible because thread they will share the code the code part is same code data everything is same. So, it is not possible to overlay one particular threads code and data.

So, as a result the exec goes normally. So, if any thread execute exec. So, entire process all the processes. So, they will be all the threads they will be overlaid as the entire processing overlaid. So, this is there.

(Refer Slide Time: 18:27)

Now, so, that is about fork and exec regarding signal handling. So, I was telling that signals are used particularly in UNIX system to notify a process that particular event has occurred. So, while executing some erroneous condition as occurred so, how the not only erroneous some imparts maybe some other information which the process want wanted to be notified.

So, that note that event has occurred then the persistent needs to be noted the process needs to be notified of the. So, how does it happen with in the context of this process in the context of threads? A signal handler is used to process signals, a signal is generated by particular event, signal is delivered to a process and signal is handled by one of the

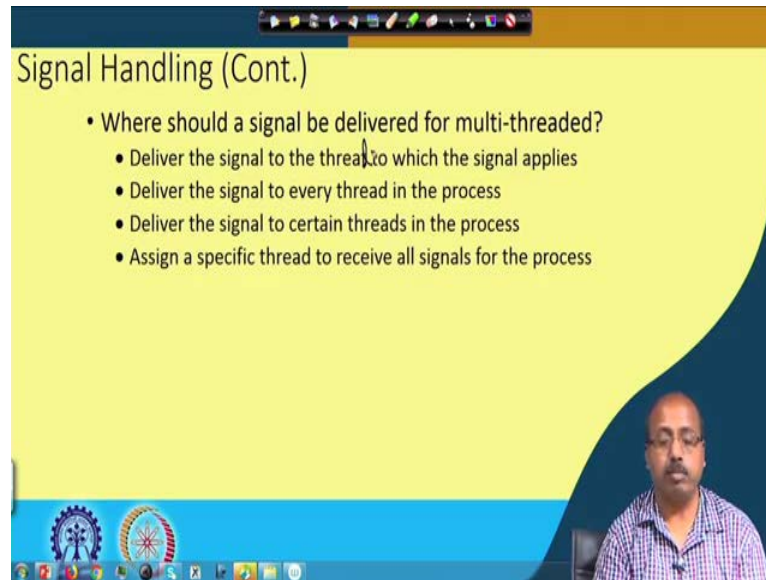
two signal handler that is the default signal handler and there can be user defined signal handler. So, these are the certain things that we have with respect to signals.

So, what you can do in a C program? So, on a UNIX system. So, you can say like you can tell like signal. So, there are different signals the we have to each signal has got a number. So, if it is say 5 comma you can talk about some routine if function 1. And function 1 is a specific routine written in the written for this particular process. So, this is the function 1. So, where this particular routine is written.

Now, what happens is that, when this signal comes signal number 5 comes which may be which is actually defined in the manual of the operating system what is this particular signals when the signal comes then the that will be inform to the process and the process will start executing this function one so, that maybe some special handling routine. So, that will be executed so, this can be there. So, signal handler that is used to process the signal. So, that is signal handler like this and signal handler it is. So, this is basically user defined signal handler that I have said.

Then there can be some default handler. So, default handler is nothing, but terminate the process that is the default action. So, when any signal comes the process is terminated the signal is handled by one of the by this default single handler. Now in general signal is deliver to the process, now how this signal is handled in a thread type of environment? So, every signal has got default handler that kernel runs when the handling the single and user defined single handed can override the default and for single threaded signal is delivered to the process. So, that is the normal situation that we have.

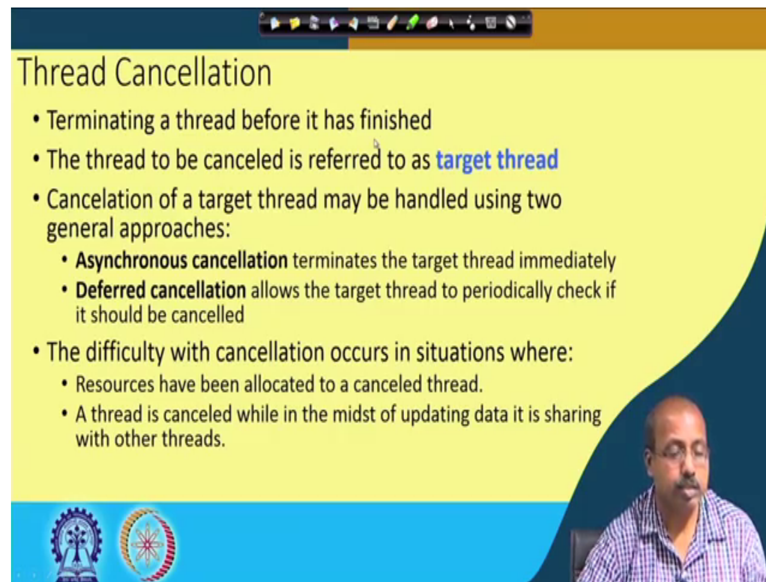
(Refer Slide Time: 21:24)



So, signals are very important because it can be used to capture many abnormal situation. So, that is fine for single threaded processes for multi threaded process what do you do? So, should we deliver signal to the thread to which the signal applies? So, I am sorry. So, this is written as thread, so, this should be thread. So, this is deliver the signal to the thread to which signal applies.

So, this is one possibility another possibilities the deliver the signal to every thread in the process that is also possible or to certain threads of the process or we can specific thread to receive all signals for the how can I have a particular thread which will be receiving all the signals of the process. So, these are the different options that we can have and which one will be done. So, that is of course, up to the waste designer.

(Refer Slide Time: 22:22)



So, you have to consult the corresponding manual to see how is it handled. Next comes the issue of thread cancellation. So, how do you terminate a thread. So, if you terminate then what happens? So, cancelation is basically terminating a thread before it has finished. A thread can be cancelled the thread that we want to cancel so, that is referred to as target thread and cancellation of a target thread may be handled using two approaches; one is asynchronous cancellation another is deferred cancellation. So, asynchronous cancellation means that it you as soon as there is a decision to terminate the thread so, it is terminated immediately.

Now how this thing happens so, that is a of course, another issue that is waste designer has to implement that so, but as soon as a decision is made to terminated it will be terminated. On the other hand this deferred cancellation so, it will allowed the target thread to periodically check if it should be cancelled. So, here the threads are I should say more well behaved thread. So, that periodic intervals of time the thread will check a flag to see whether these it has been asked to terminate. So, maybe there are some thread that was initiated previously, but it is found that the action of that thread is no more required some other situation has occurred and so, that is not there.

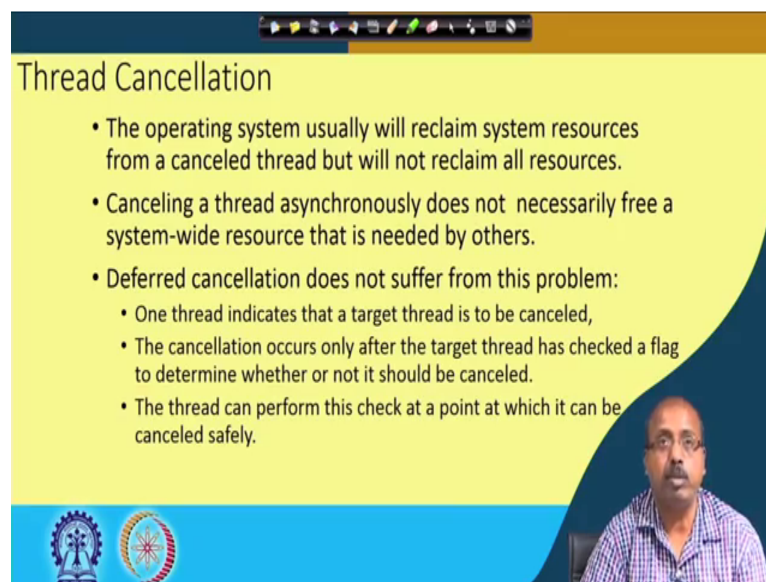
So, ideally I should be able to set a flag and this previous thread which was doing the operation suppose which was supposed to do the operation, say it will periodically check that flag and if there flag is set it will not to the operation or rather terminate or get

cancelled. So, this deferred cancellation. So, this is this will allow the target thread to periodically check if it is a bound to give it to be cancelled.

Now, difficulty comes because there will be resources associated with the thread. So, the resources that is there with the threads of what happens to those resources when the thread gets cancelled? And as you know that resource handling resource management is done at the process level. So, if you a process may have hundreds threads out of which one thread is getting cancelled or terminated.

Now what happens to the resources held by these threads? So, if you release this resources and other threads is also suffered. So, they have; so, resources have been allocated to a cancel threads. So, a thread cancelled while in the midst of updating data it is sharing with other threads. So, this is that difficulty that can come.

(Refer Slide Time: 25:00)

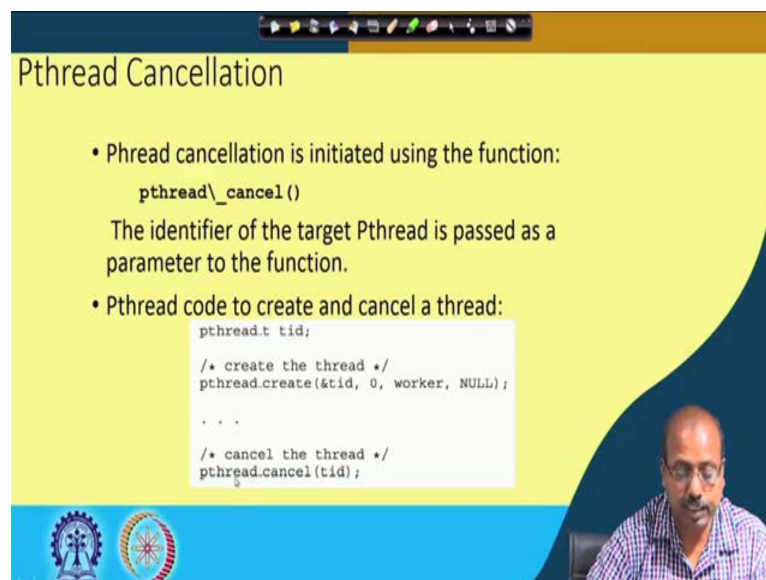


So, how to handle it; how to handle this termination is, it has to be done more carefully. The operating system usually will be reclaim the system resources. So, I cancel thread, but will not reclaim all resources. So, if the waste finds that there are some resources which are held by only this particular thread so, they will be reclaimed, but not all resources. So, the resources that are held by other threads. Cancelling a asynchronously does not necessarily free a system wide resource that is needed by others.

So, this is that come this is the implication of that. So if you are cancelling a thread so, it does not mean synchronously. So, it does not mean that it will receive a free all the resources. Deferred cancellation of course, has got the advantage that it does not suffer from the problem because once it is indicated that a thread be terminated the to be or to be cancelled, the cancellation occurs only after the thread has checked a flag and determine whether or not it should be cancelled. So, that thread can check like what are the now how to cancel it safely.

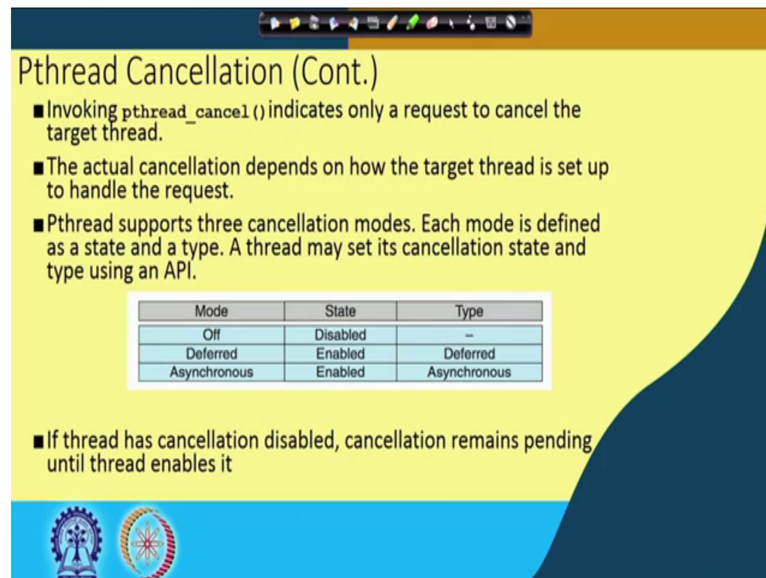
So, it can find out like what are the resources it can release that will not hamper other threads and accordingly it can take place. So, this thread cancellation of this deferred cancellation is if it safe I should say as for this resource handling is concerned.

(Refer Slide Time: 26:23)



So this is how this thread cancellation takes place. So, Pthread cancellation is initiated using the function pthread cancel and the identifier for target Pthread is passed as a parameter to the function. So, this is a routine. So, pthread creates, so, this t id was thread id for cancelling the threads I should say pthread cancel tid. So, that way this pthread will be informed that there is a cancellation.

(Refer Slide Time: 26:51)



Now, as far as cancellation is concerned. So, this is only a request to cancel the target thread. So, this is basically a deferred cancellation type of approach that is there in Pthread, the actual cancellation depends how the target thread is setup to handle the request. So, these are the different modes. So, the mode maybe of so; that means, that this cancellation cannot be done. So, thread will never cannot be cancelled that is the state is disabled.

The mode can be deferred so, that way this state is enabled and the type is deferred. So, this thread will periodically check whether the cancellation request has come and then accordingly it will cancel it or it may be asynchronous. So, if it is a the state is enabled. So, if the mode of the thread is asynchronous cancellation then it will be cancelling it immediately. So, this way we can create a P thread specification says that you can create threads with different types of modes and depending upon the mode that you have. So, this cancellation will go through different types of situations.

So, this is the as again the same thing that is this is only by only a specification. So, how this thing is implemented in an operating system that is an issue and that is up to the operating system designer to take a decision.