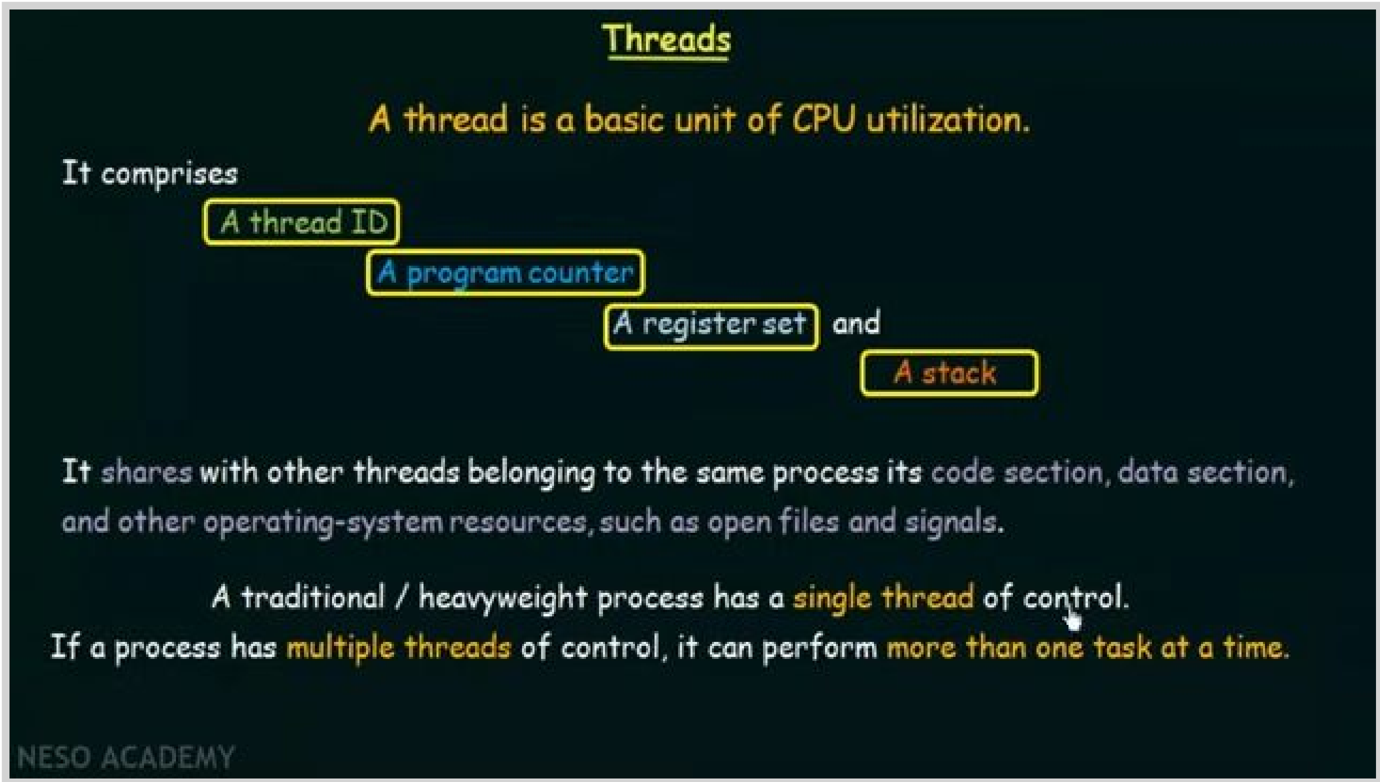
←

PDF

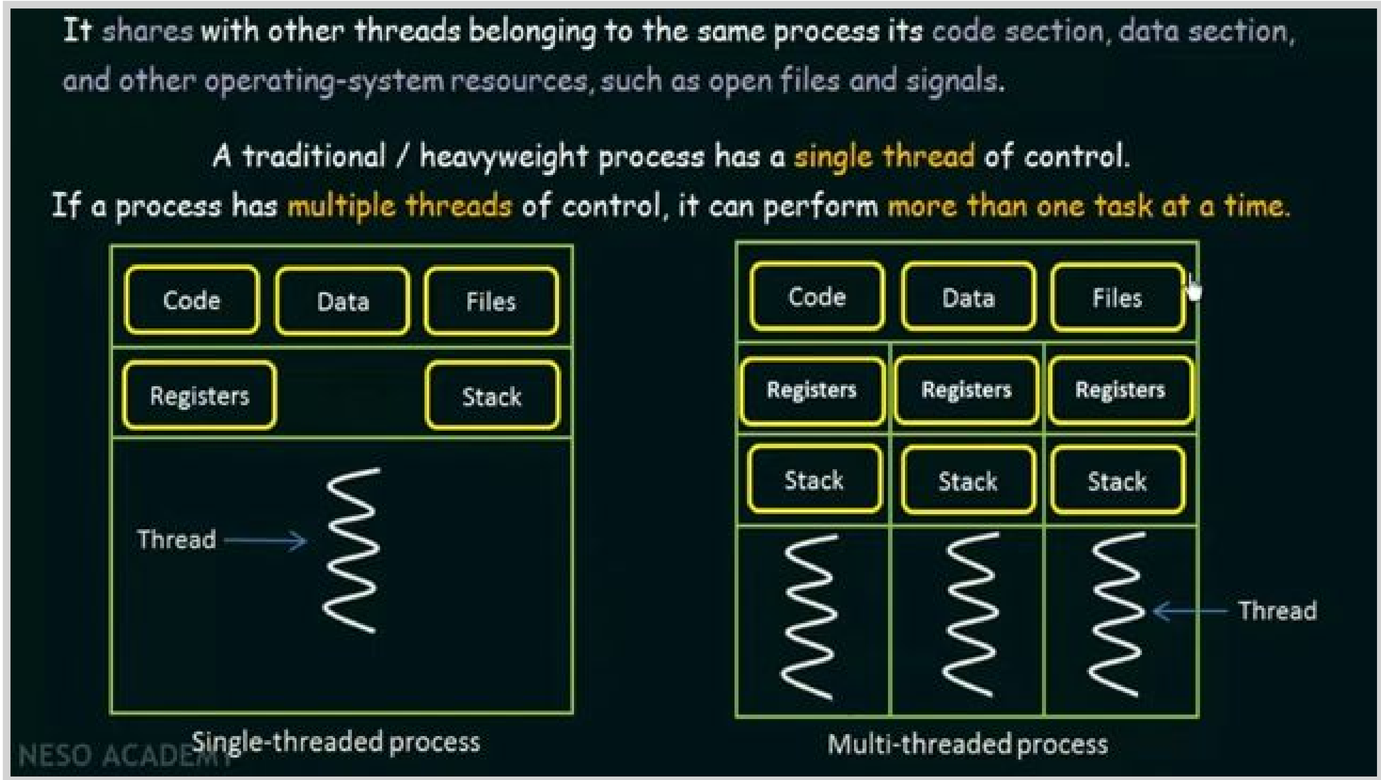
PDF

[01:41]

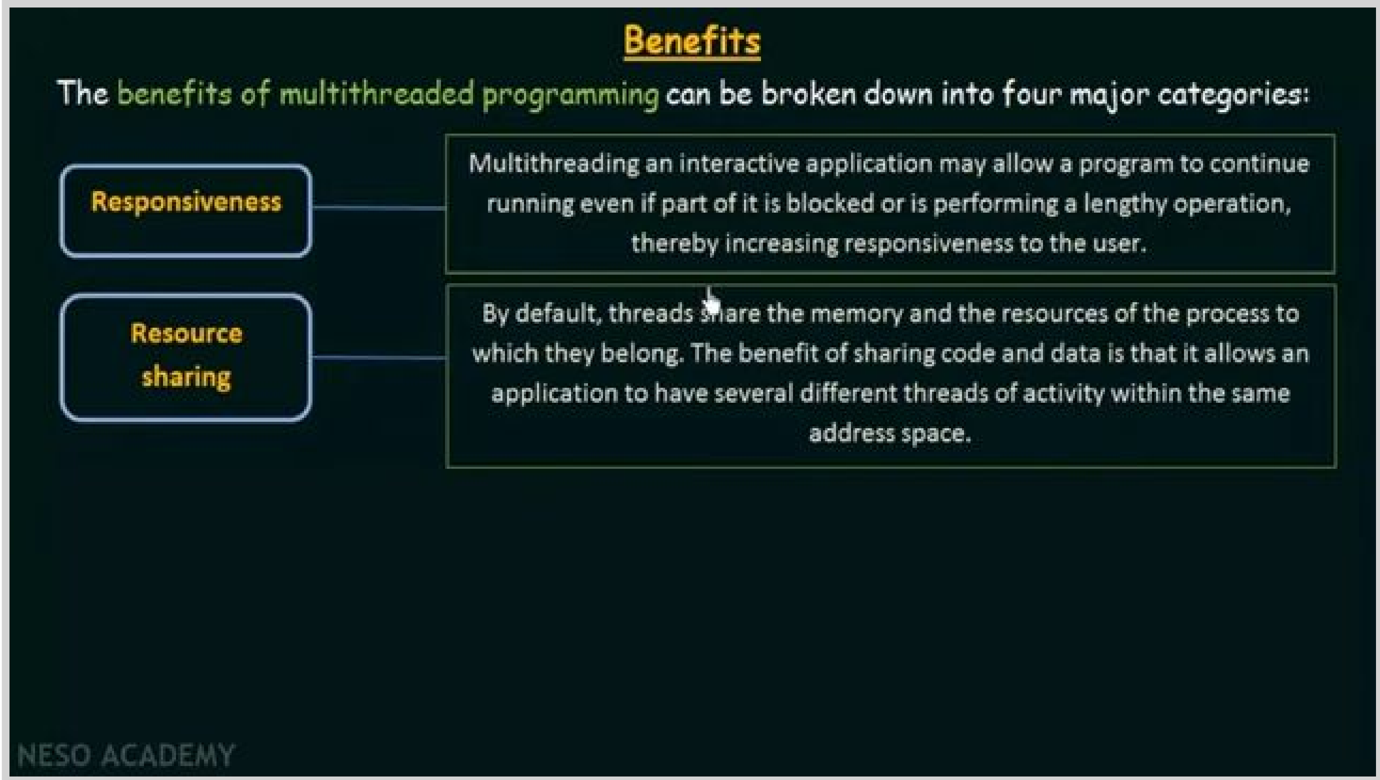


[03:08]

←


It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
A traditional / heavyweight process has a single thread of control.
If a process has multiple threads of control, it can perform more than one task at a time.
Code
Data
Files
Registers
Stack
Thread
Code
Data
Files
Registers
Registers
Registers
Stack
Stack
Stack
Thread
NESO ACADEMY
Single-threaded process
Multi-threaded process

[09:22]

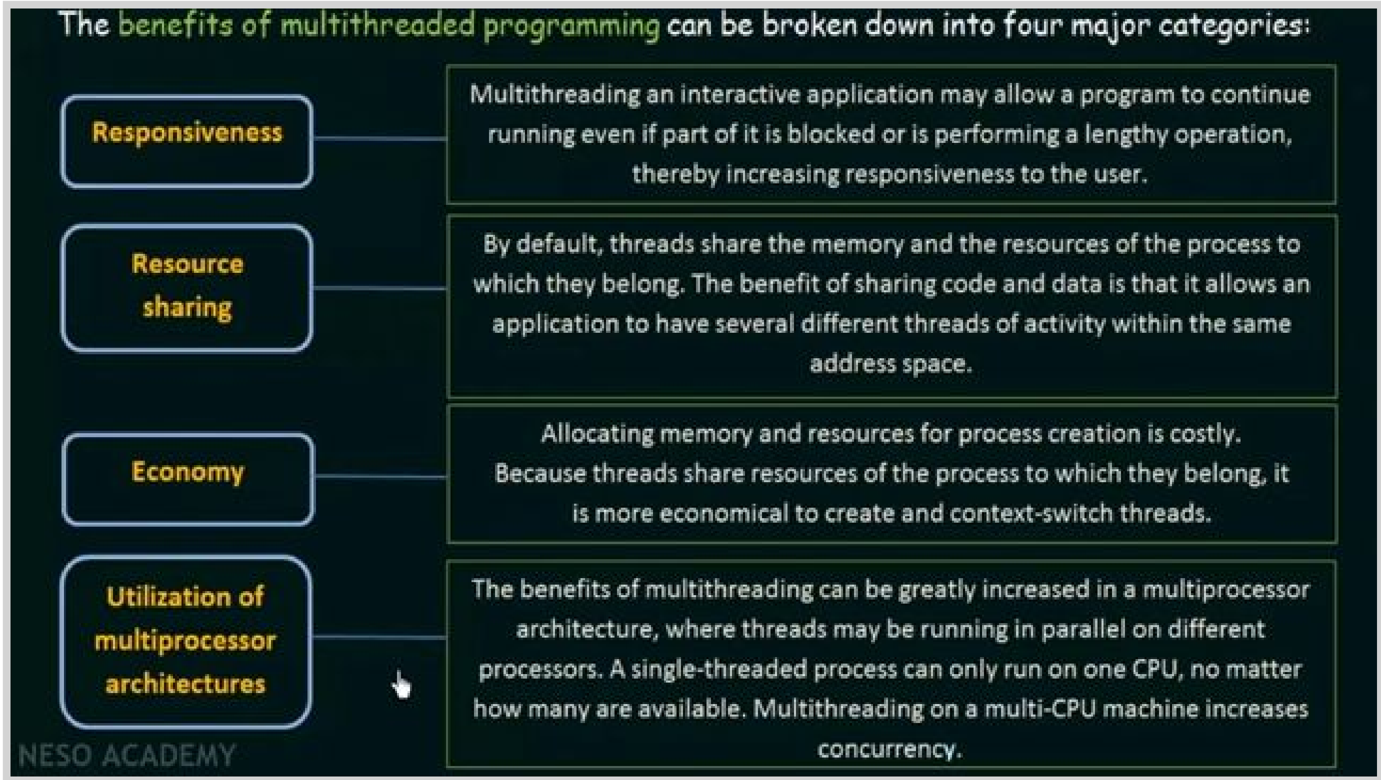

Benefits
The benefits of multithreaded programming can be broken down into four major categories:
Responsiveness
Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
Resource sharing
By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
NESO ACADEMY

PDF

PDF

[12:51](#)

←

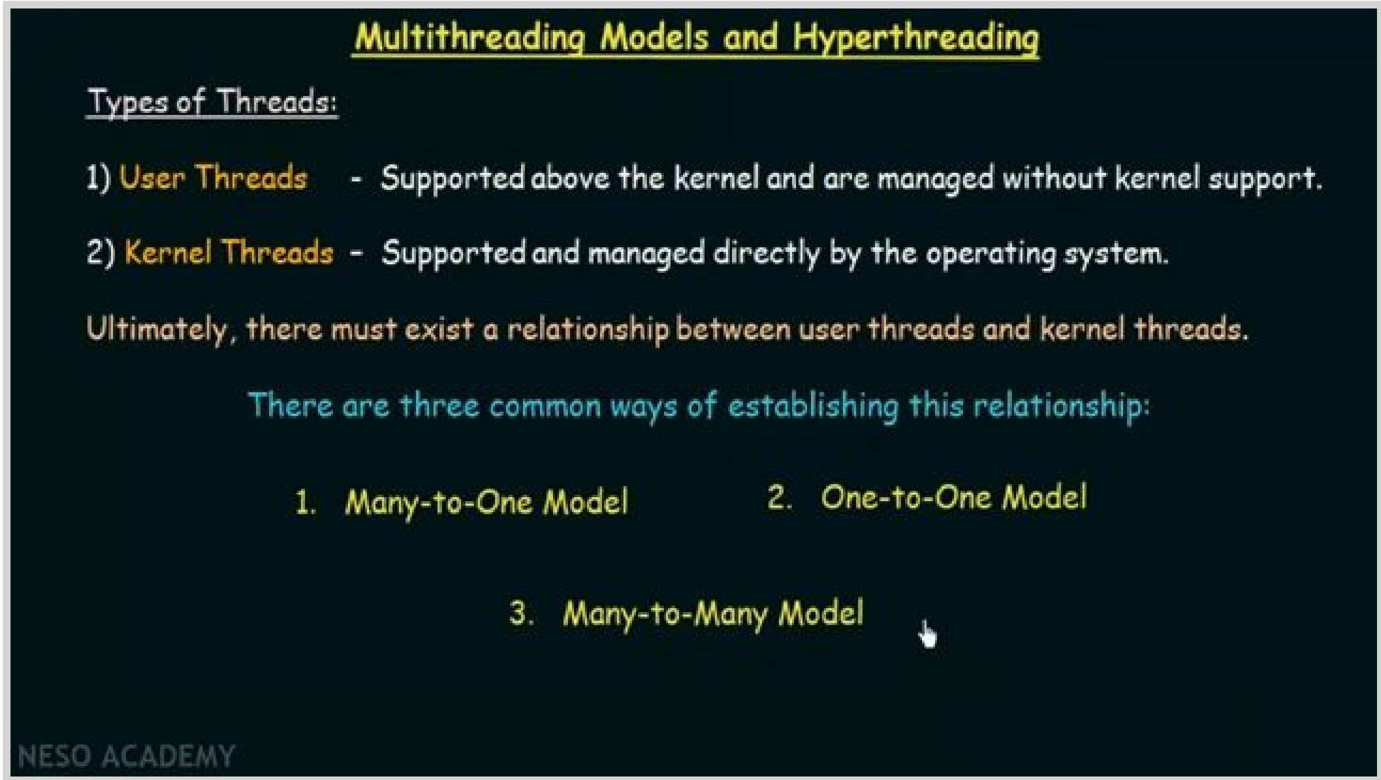

The benefits of multithreaded programming can be broken down into four major categories:
Responsiveness
Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
Resource sharing
By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
Economy
Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
Utilization of multiprocessor architectures
The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases concurrency.
NESO ACADEMY

PDF

PDF

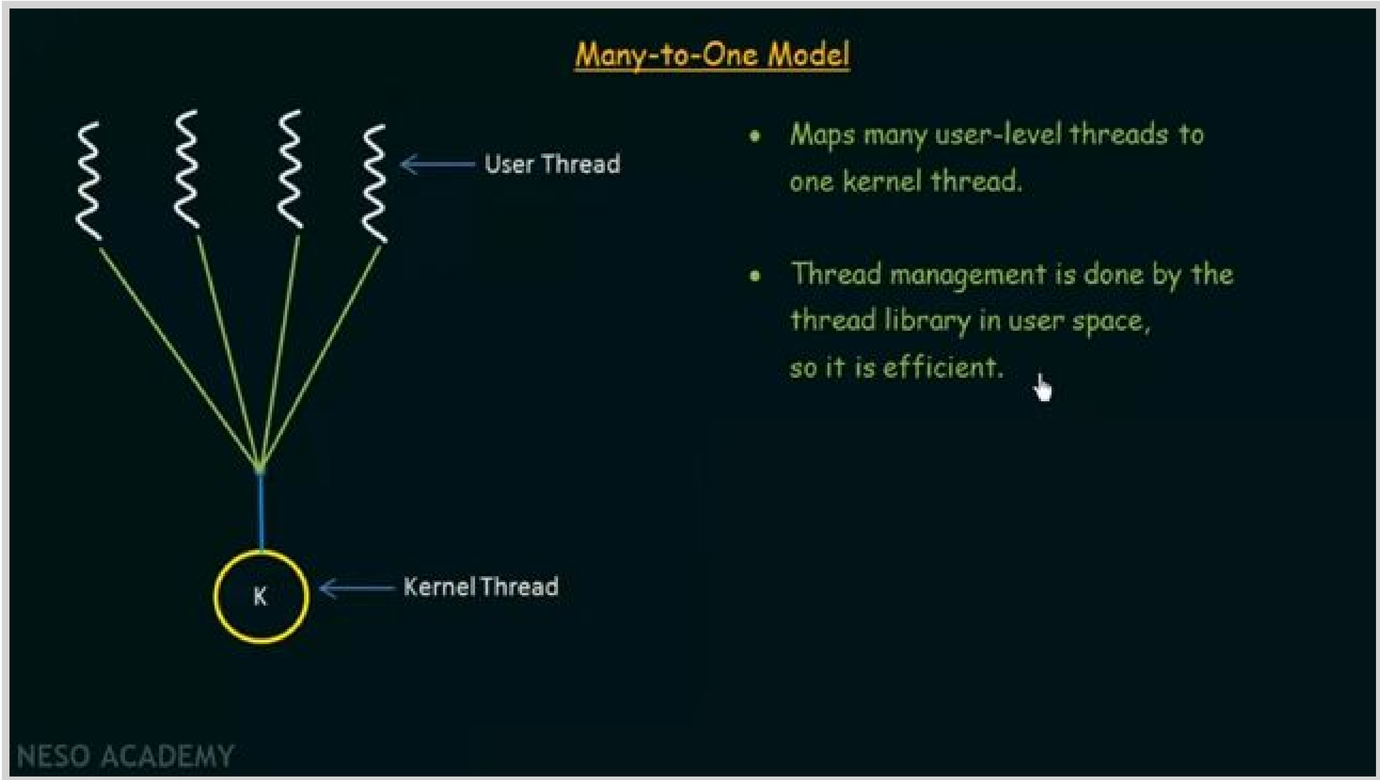
[02:07]

←


Multithreading Models and Hyperthreading
Types of Threads:
1) User Threads - Supported above the kernel and are managed without kernel support.
2) Kernel Threads - Supported and managed directly by the operating system.
Ultimately, there must exist a relationship between user threads and kernel threads.
There are three common ways of establishing this relationship:
1. Many-to-One Model
2. One-to-One Model
3. Many-to-Many Model
NESO ACADEMY

[03:12]

←

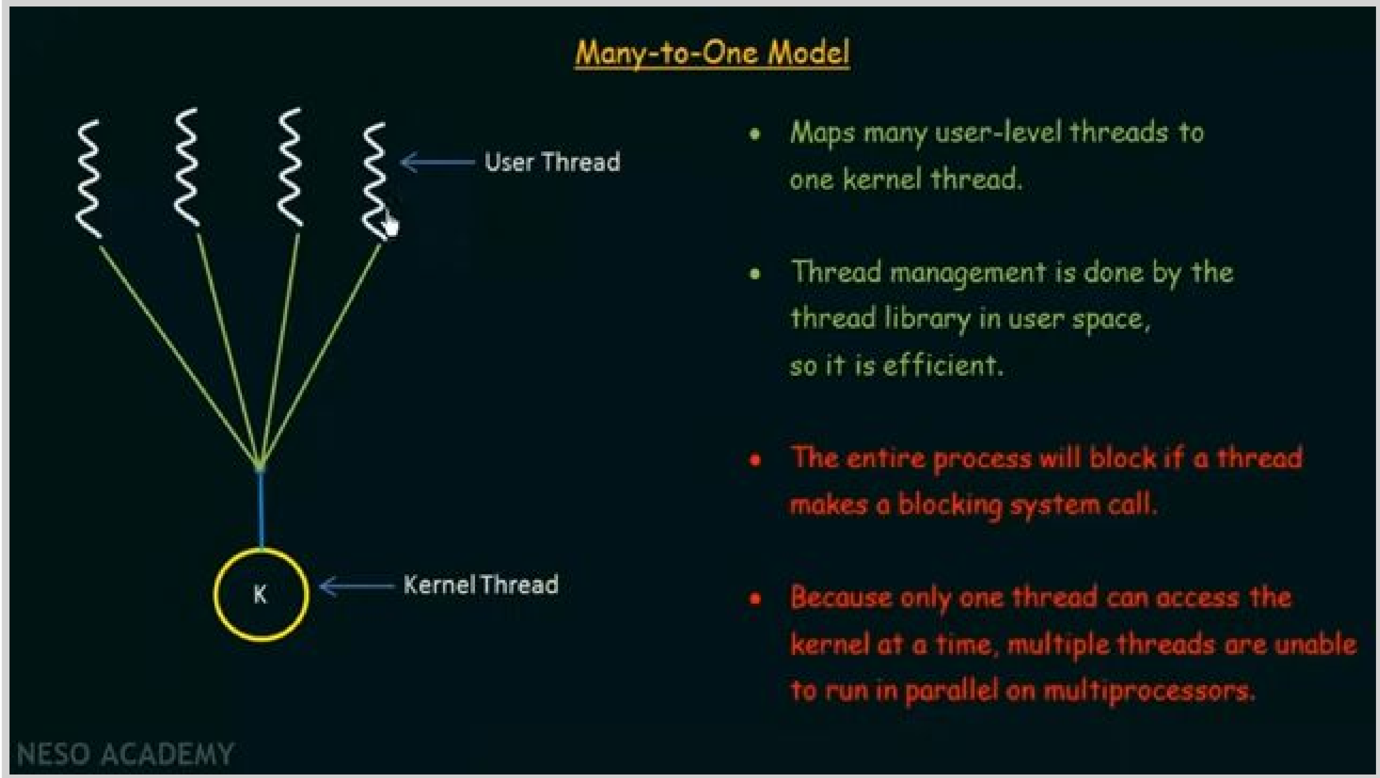

Many-to-One Model
User Thread
K
Kernel Thread
• Maps many user-level threads to one kernel thread.
• Thread management is done by the thread library in user space, so it is efficient.
NESO ACADEMY

PDF

PDF

[03:34]

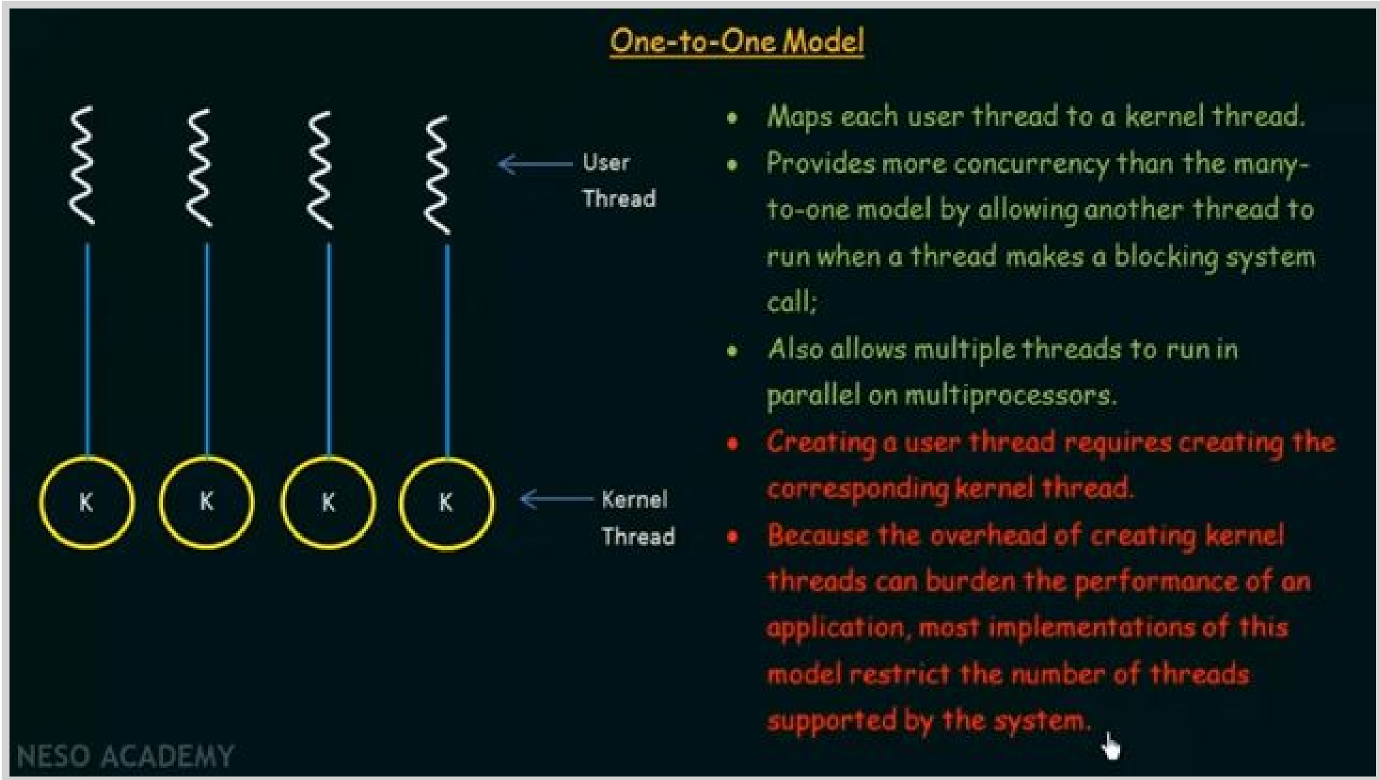
←


Many-to-One Model
User Thread
K
Kernel Thread
• Maps many user-level threads to one kernel thread.
• Thread management is done by the thread library in user space, so it is efficient.
• The entire process will block if a thread makes a blocking system call.
• Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.
NESO ACADEMY

PDF
PDF

[07:48]

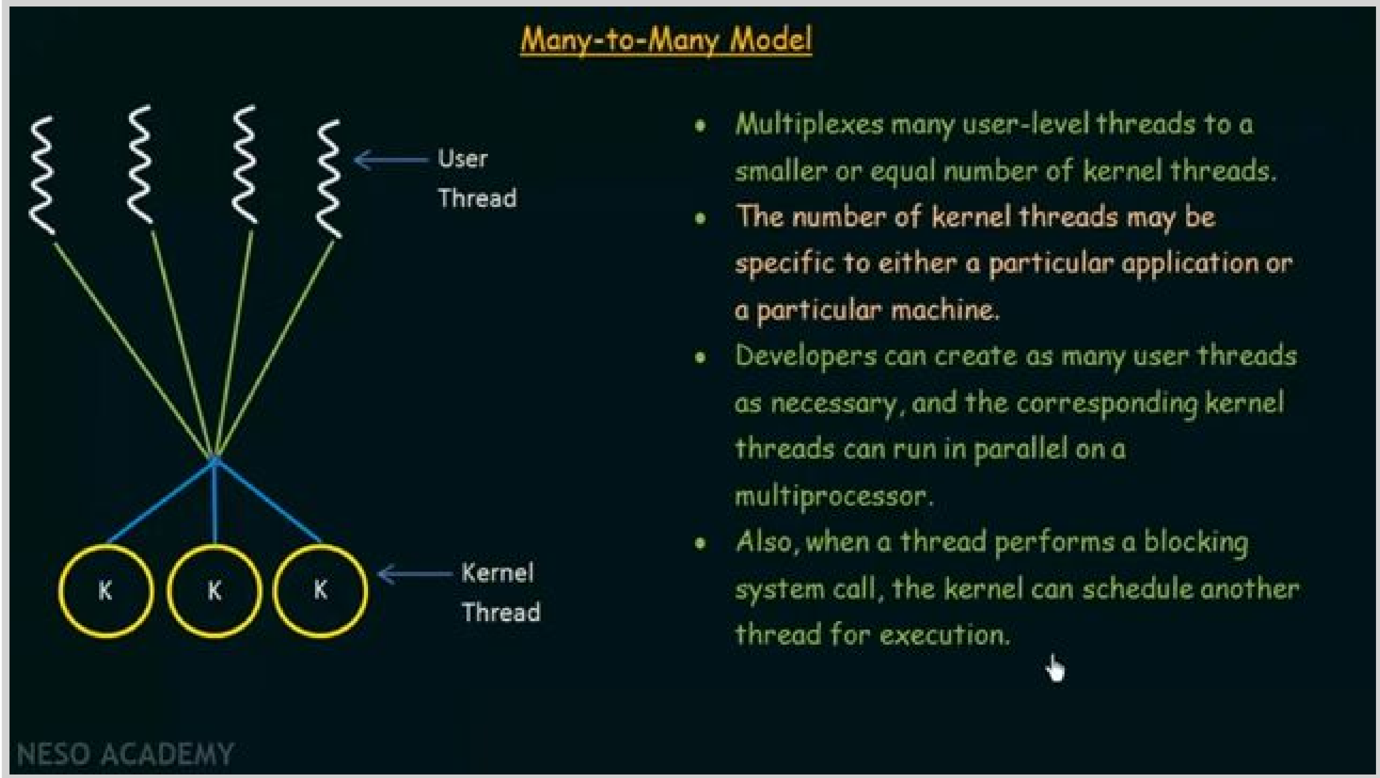
←


One-to-One Model
User Thread
Kernel Thread
K
K
K
K
Maps each user thread to a kernel thread.
Provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call;
Also allows multiple threads to run in parallel on multiprocessors.
Creating a user thread requires creating the corresponding kernel thread.
Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of threads supported by the system.
NESO ACADEMY

PDF
PDF

[10:29]

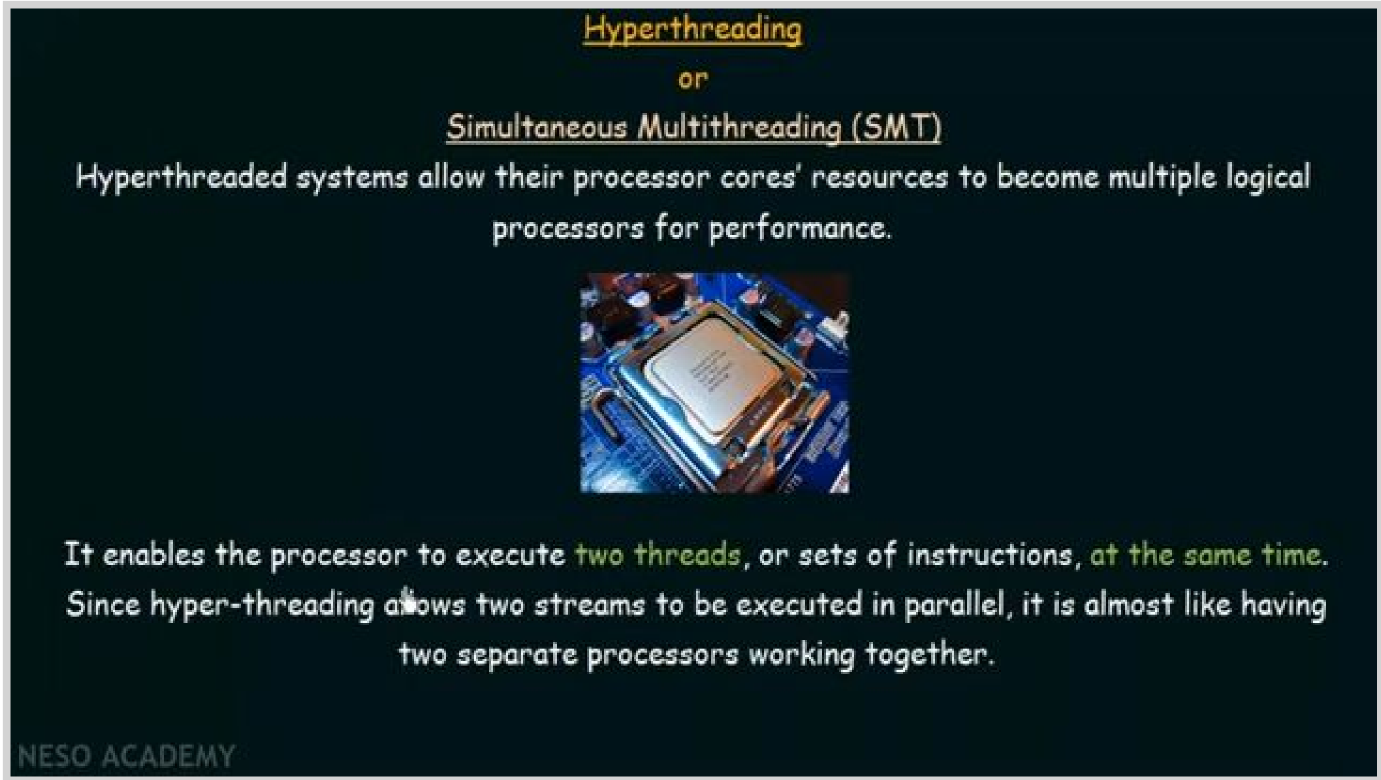
←


Many-to-Many Model
User
Thread
K
K
K
Kernel
Thread
• Multiplexes many user-level threads to a smaller or equal number of kernel threads.
• The number of kernel threads may be specific to either a particular application or a particular machine.
• Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
• Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.
NESO ACADEMY

PDF
PDF

[14:05]()

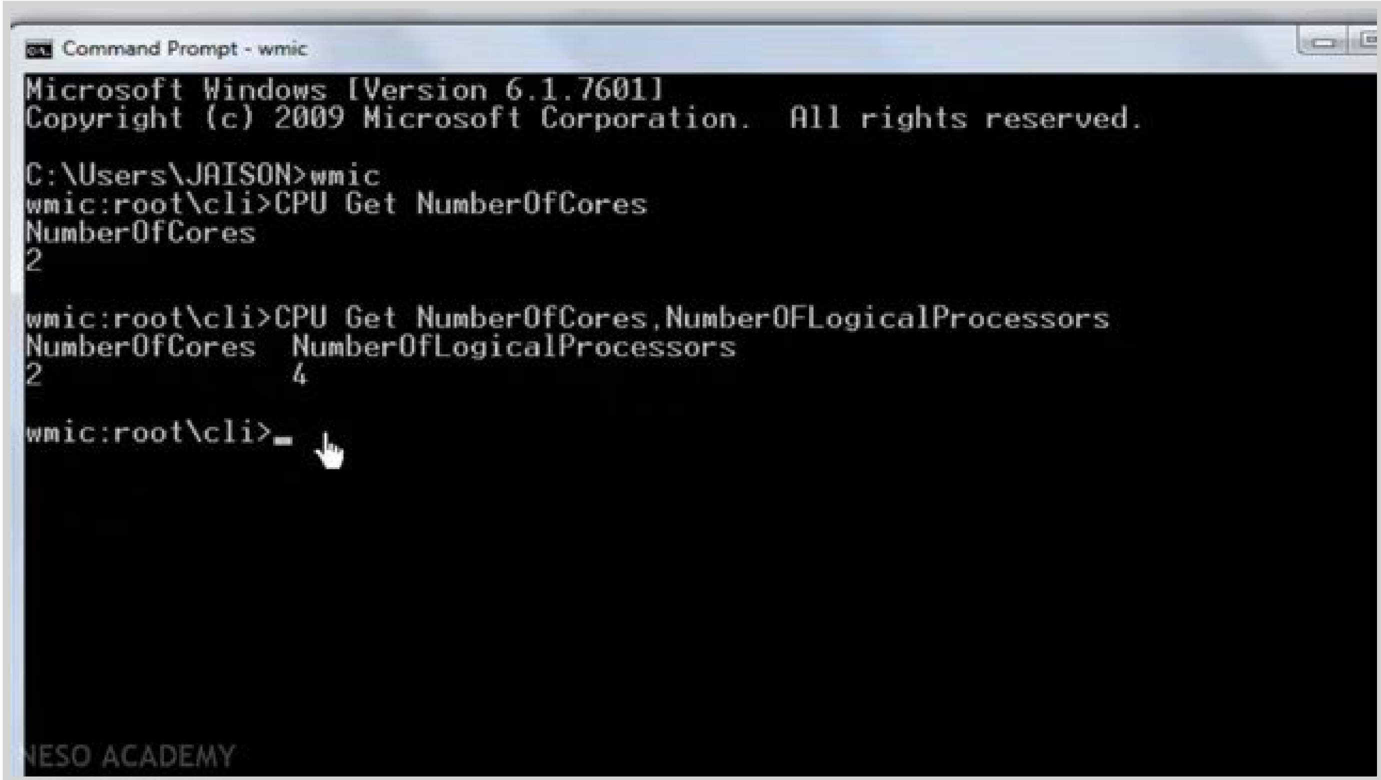
←


Hyperthreading
or
Simultaneous Multithreading (SMT)
Hyperthreaded systems allow their processor cores' resources to become multiple logical processors for performance.
It enables the processor to execute two threads, or sets of instructions, at the same time.
Since hyper-threading allows two streams to be executed in parallel, it is almost like having two separate processors working together.
NESO ACADEMY

PDF

PDF

[17:11]

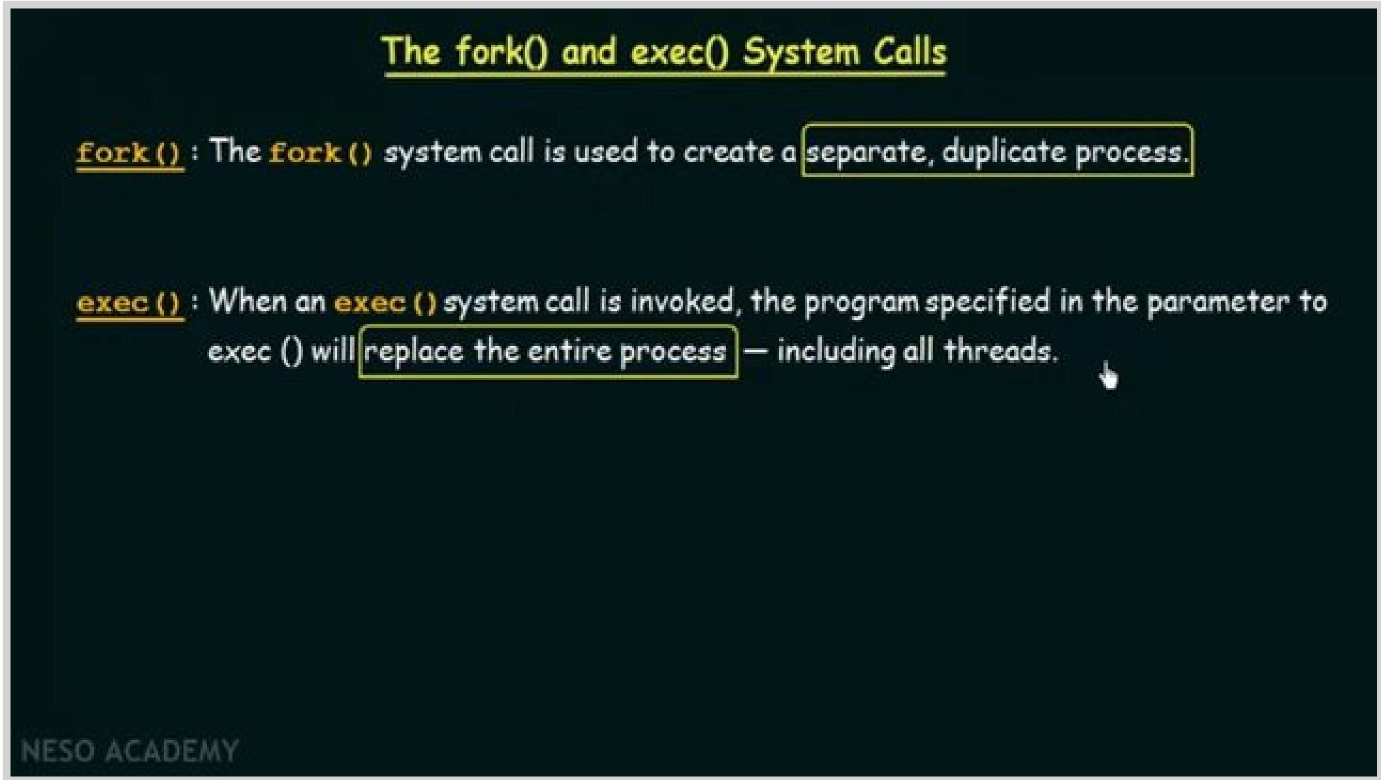

Command Prompt - wmic
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\JAISON>wmic
wmic:root\cli>CPU Get NumberOfCores
NumberOfCores
2
wmic:root\cli>CPU Get NumberOfCores,NumberOFLogicalProcessors
NumberOfCores NumberOfLogicalProcessors
2 4
wmic:root\cli>
NESO ACADEMY

PDF

PDF

[03:31]

←

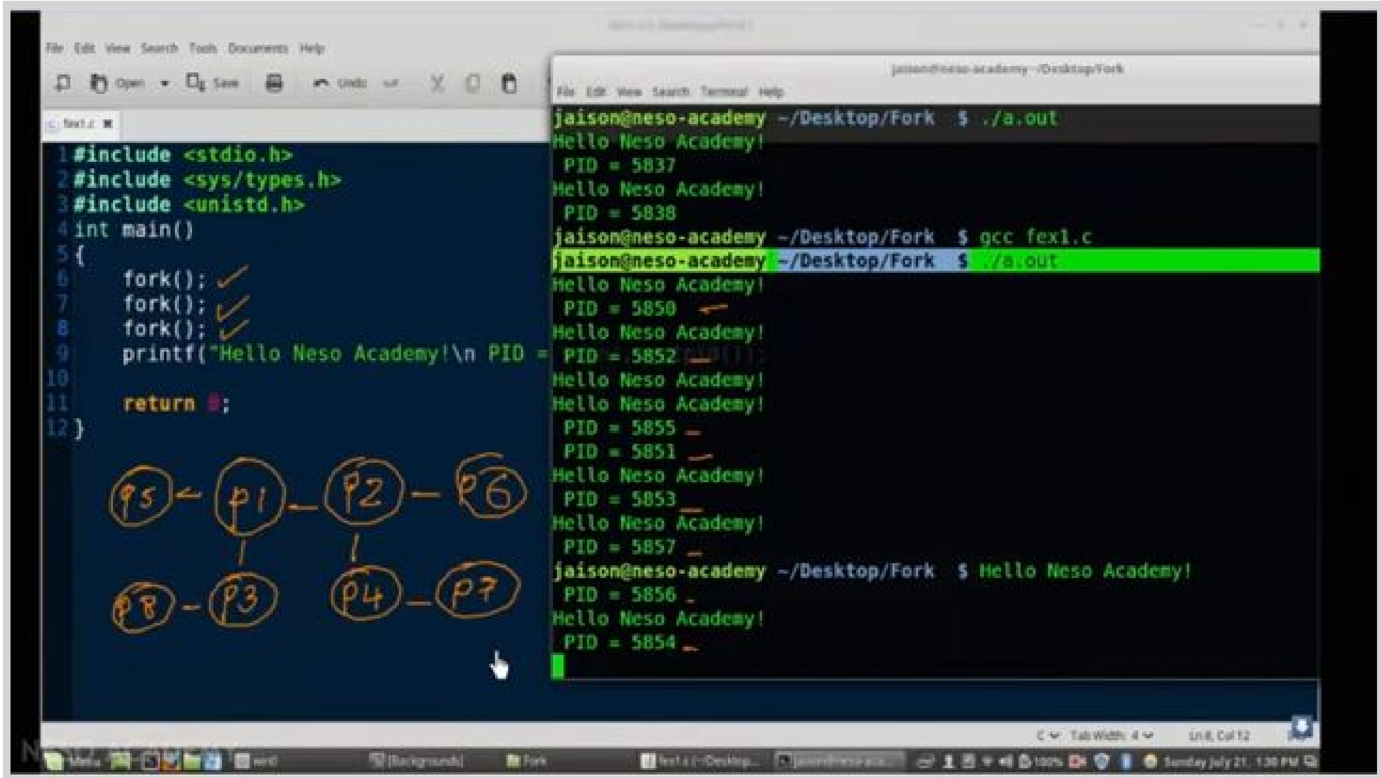


PDF
PDF

ccE` 1q `11

[12:06]

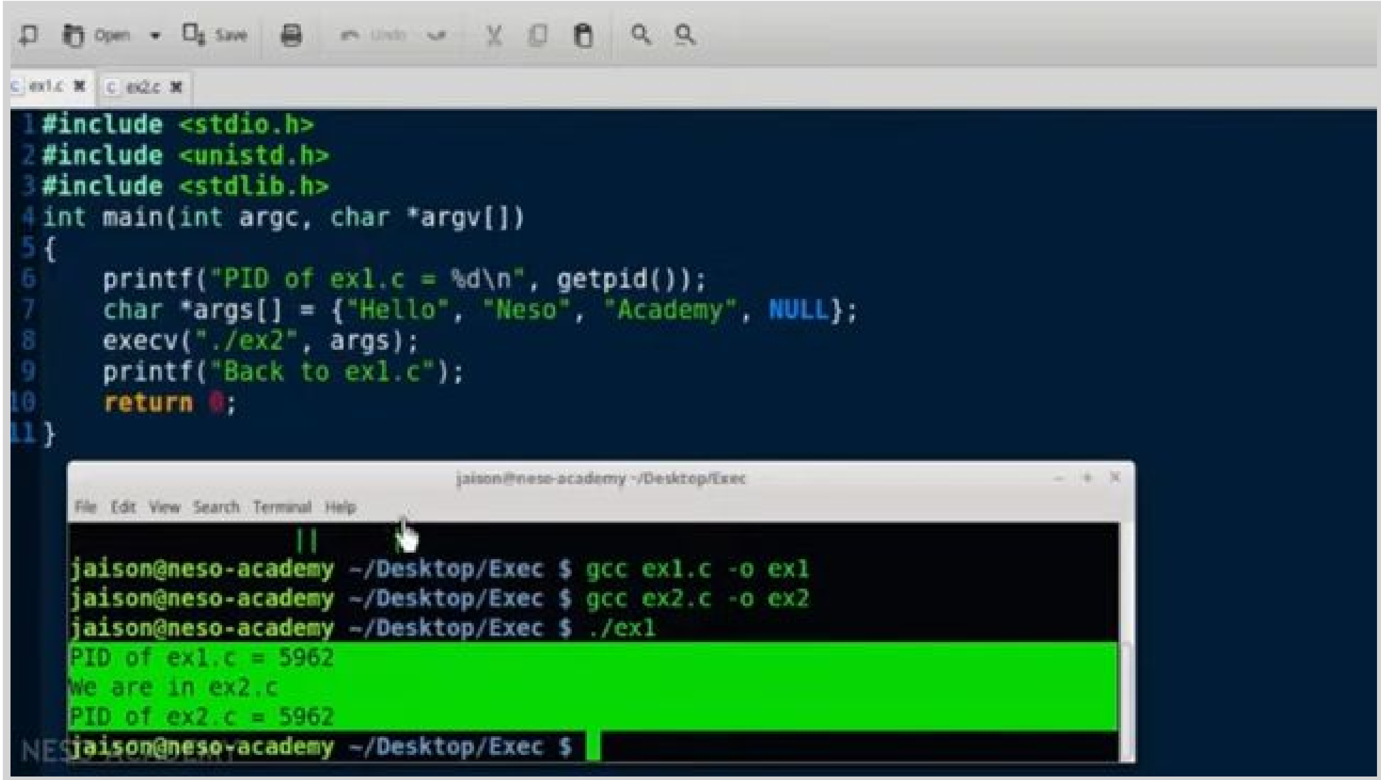
←


#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
fork();
fork();
fork();
printf("Hello Neso Academy!\n PID =
return 0;
}
P5 P1 P2 P6
P8 P3 P4 P7
jaison@neso-academy ~/Desktop/Fork $ ./a.out
Hello Neso Academy!
PID = 5837
Hello Neso Academy!
PID = 5838
jaison@neso-academy ~/Desktop/Fork $ gcc fex1.c
jaison@neso-academy ~/Desktop/Fork $ ./a.out
Hello Neso Academy!
PID = 5850
Hello Neso Academy!
PID = 5852
Hello Neso Academy!
Hello Neso Academy!
PID = 5855
PID = 5851
Hello Neso Academy!
PID = 5853
Hello Neso Academy!
PID = 5857
jaison@neso-academy ~/Desktop/Fork $ Hello Neso Academy!
PID = 5856
Hello Neso Academy!
PID = 5854

[17:02]

PDF
PDF

←

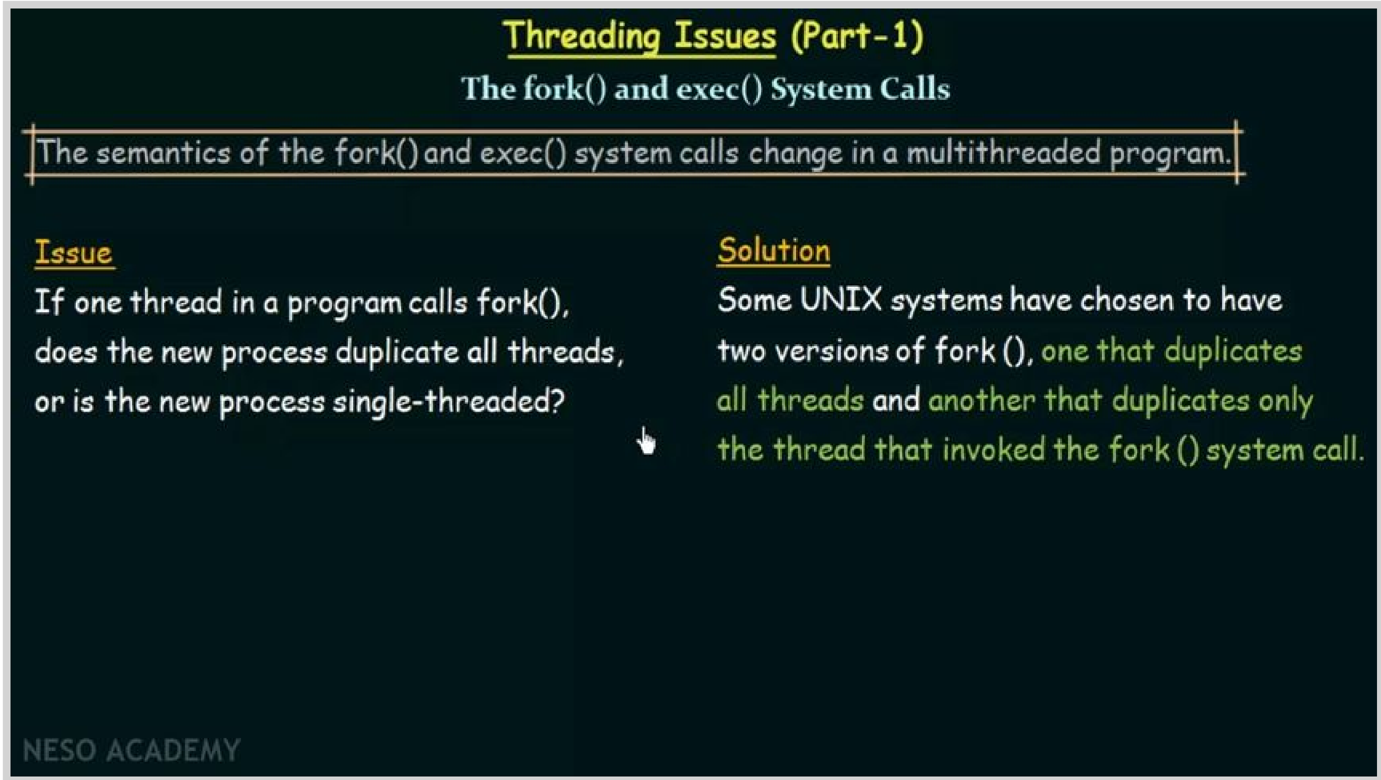

Open
Save
Undo
ex1.c
ex2.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
printf("PID of ex1.c = %d\n", getpid());
char *args[] = {"Hello", "Neso", "Academy", NULL};
execv("./ex2", args);
printf("Back to ex1.c");
return 0;
}
jaison@neso-academy ~/Desktop/Exec
File Edit View Search Terminal Help
jaison@neso-academy ~/Desktop/Exec $ gcc ex1.c -o ex1
jaison@neso-academy ~/Desktop/Exec $ gcc ex2.c -o ex2
jaison@neso-academy ~/Desktop/Exec $ ./ex1
PID of ex1.c = 5962
We are in ex2.c
PID of ex2.c = 5962
jaison@neso-academy ~/Desktop/Exec $

PDF

PDF

[02:35]

←

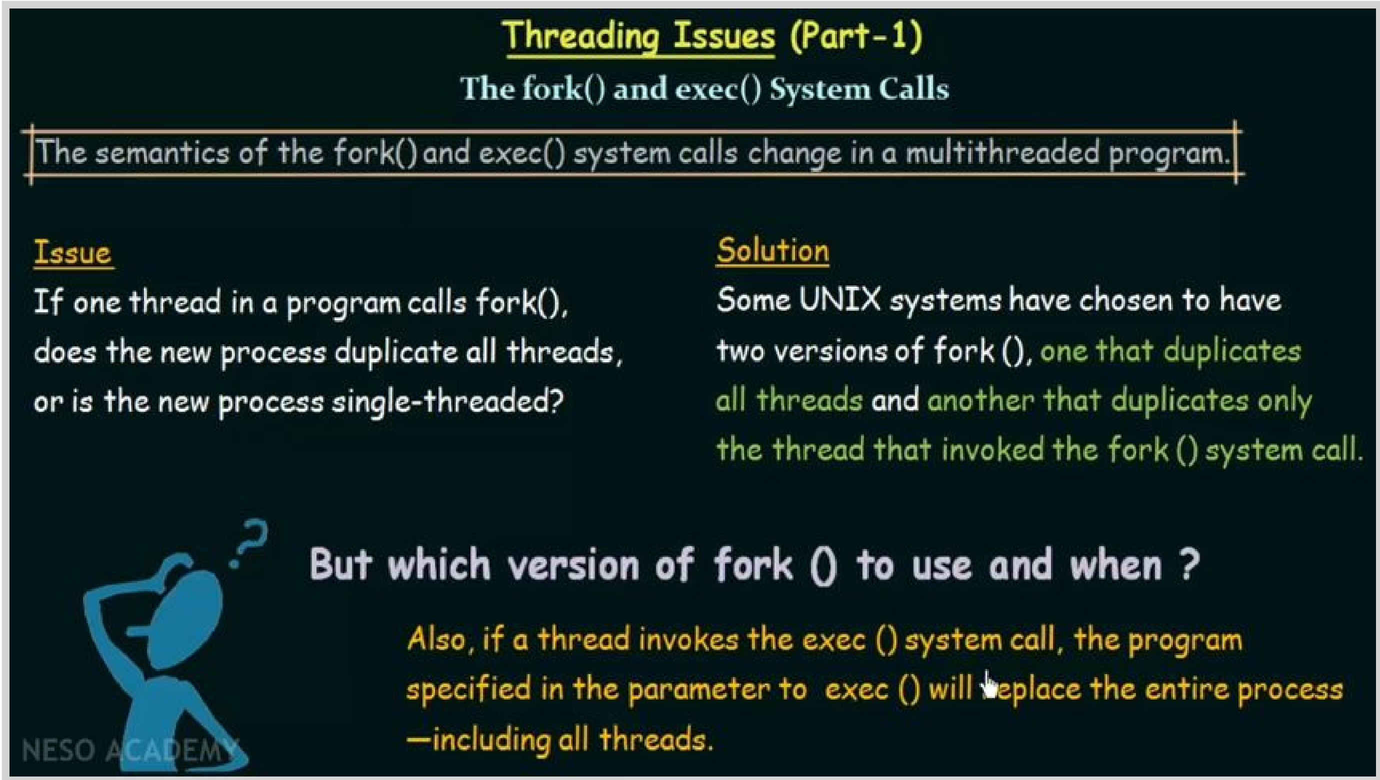


PDF
PDF

[03:52]

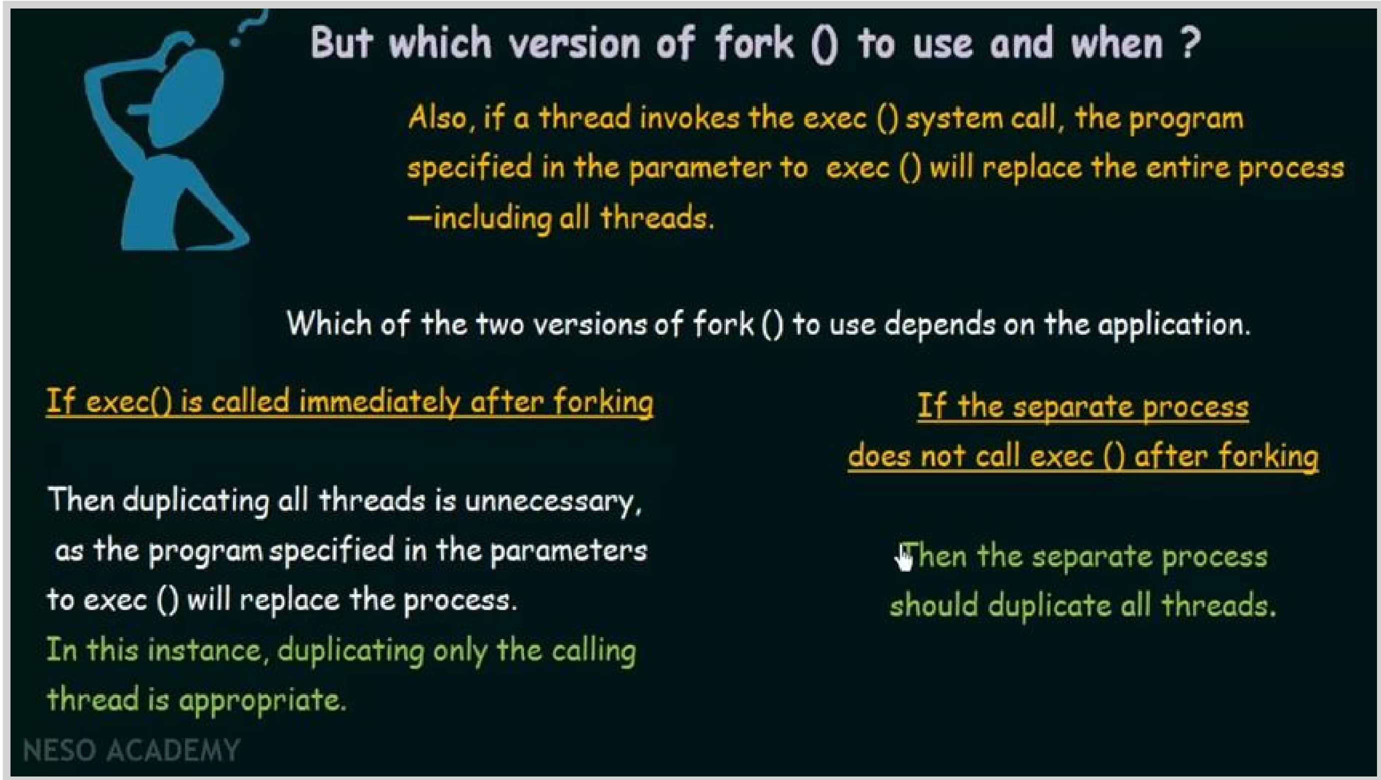
[04:20]

←


Threading Issues (Part-1)
The fork() and exec() System Calls
The semantics of the fork() and exec() system calls change in a multithreaded program.
Issue
If one thread in a program calls fork(), does the new process duplicate all threads, or is the new process single-threaded?
Solution
Some UNIX systems have chosen to have two versions of fork (), one that duplicates all threads and another that duplicates only the thread that invoked the fork () system call.
But which version of fork () to use and when ?
Also, if a thread invokes the exec () system call, the program specified in the parameter to exec () will replace the entire process —including all threads.
NESO ACADEMY

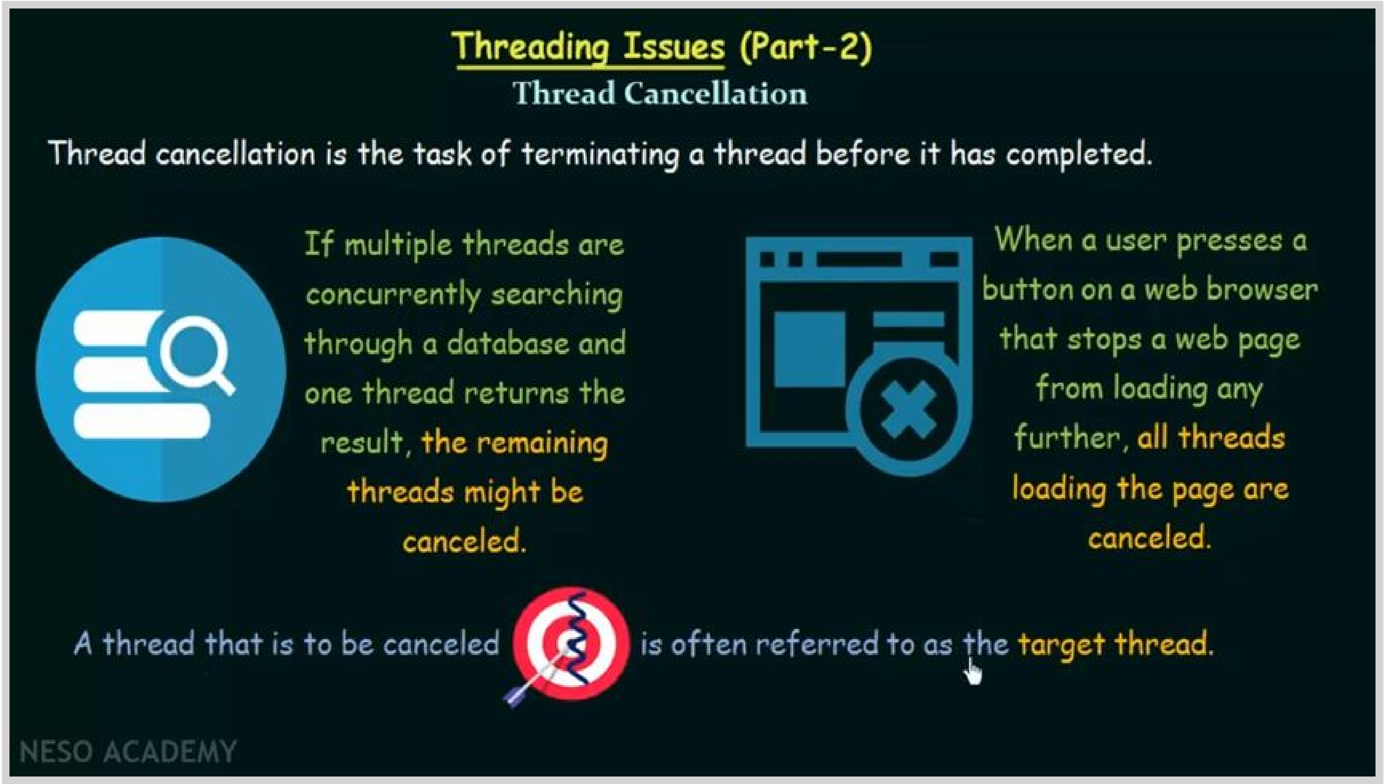
PDF
PDF

[08:04]


But which version of fork () to use and when ?
Also, if a thread invokes the exec () system call, the program specified in the parameter to exec () will replace the entire process —including all threads.
Which of the two versions of fork () to use depends on the application.
If exec() is called immediately after forking
Then duplicating all threads is unnecessary, as the program specified in the parameters to exec () will replace the process.
In this instance, duplicating only the calling thread is appropriate.
If the separate process does not call exec () after forking
Then the separate process should duplicate all threads.
NESO ACADEMY

[01:34]()

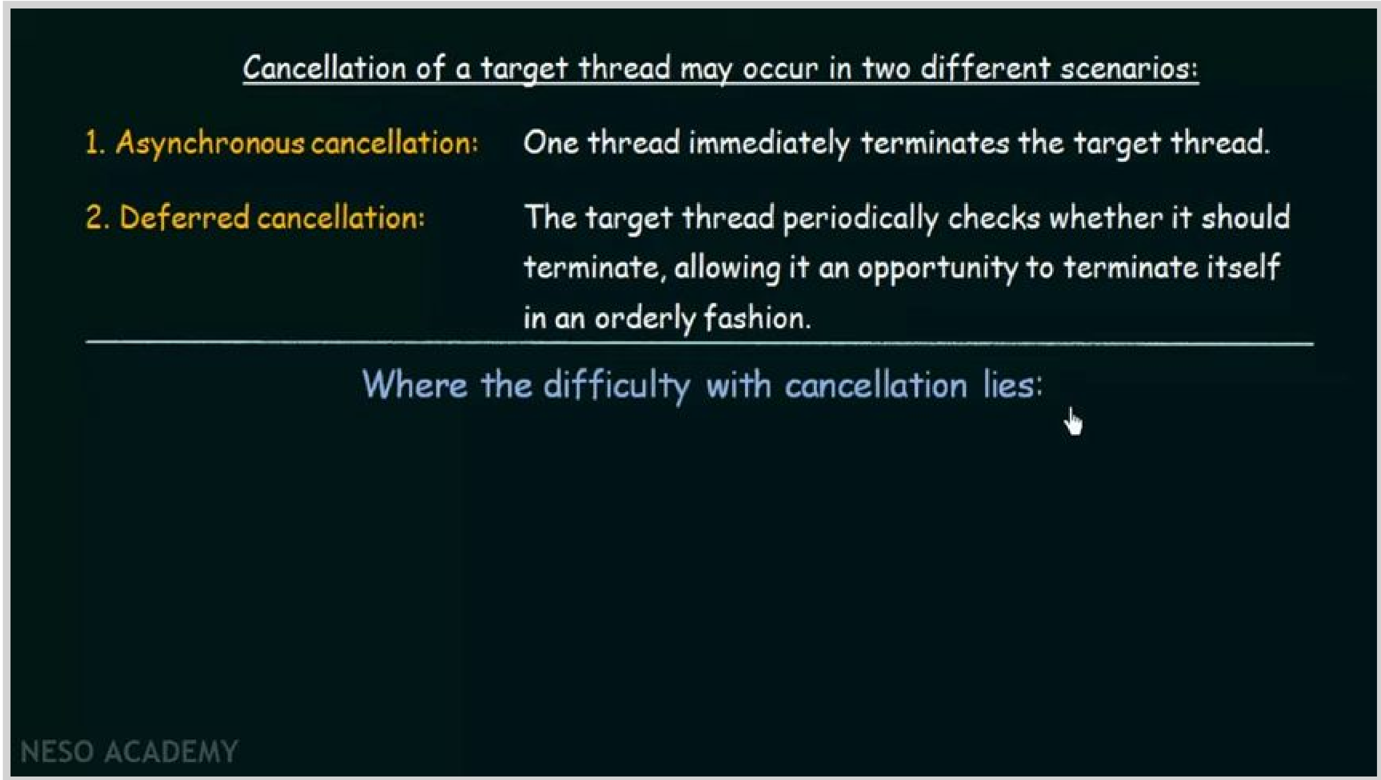
[03:11]()


Threading Issues (Part-2)
Thread Cancellation
Thread cancellation is the task of terminating a thread before it has completed.
If multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled.
When a user presses a button on a web browser that stops a web page from loading any further, all threads loading the page are canceled.
A thread that is to be canceled is often referred to as the target thread.
NESO ACADEMY

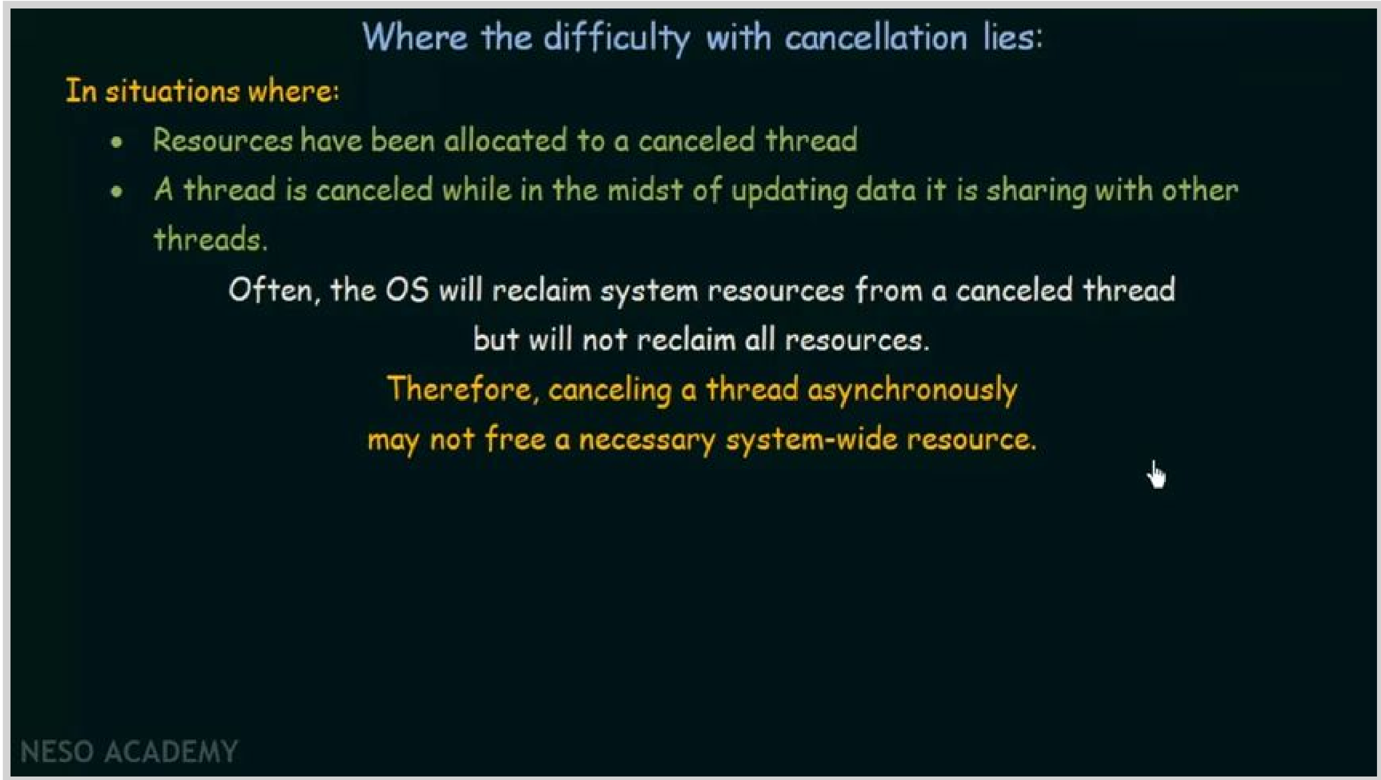
[05:47]

PDF
PDF

←


Cancellation of a target thread may occur in two different scenarios:
1. Asynchronous cancellation:
One thread immediately terminates the target thread.
2. Deferred cancellation:
The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.
Where the difficulty with cancellation lies:
NESO ACADEMY

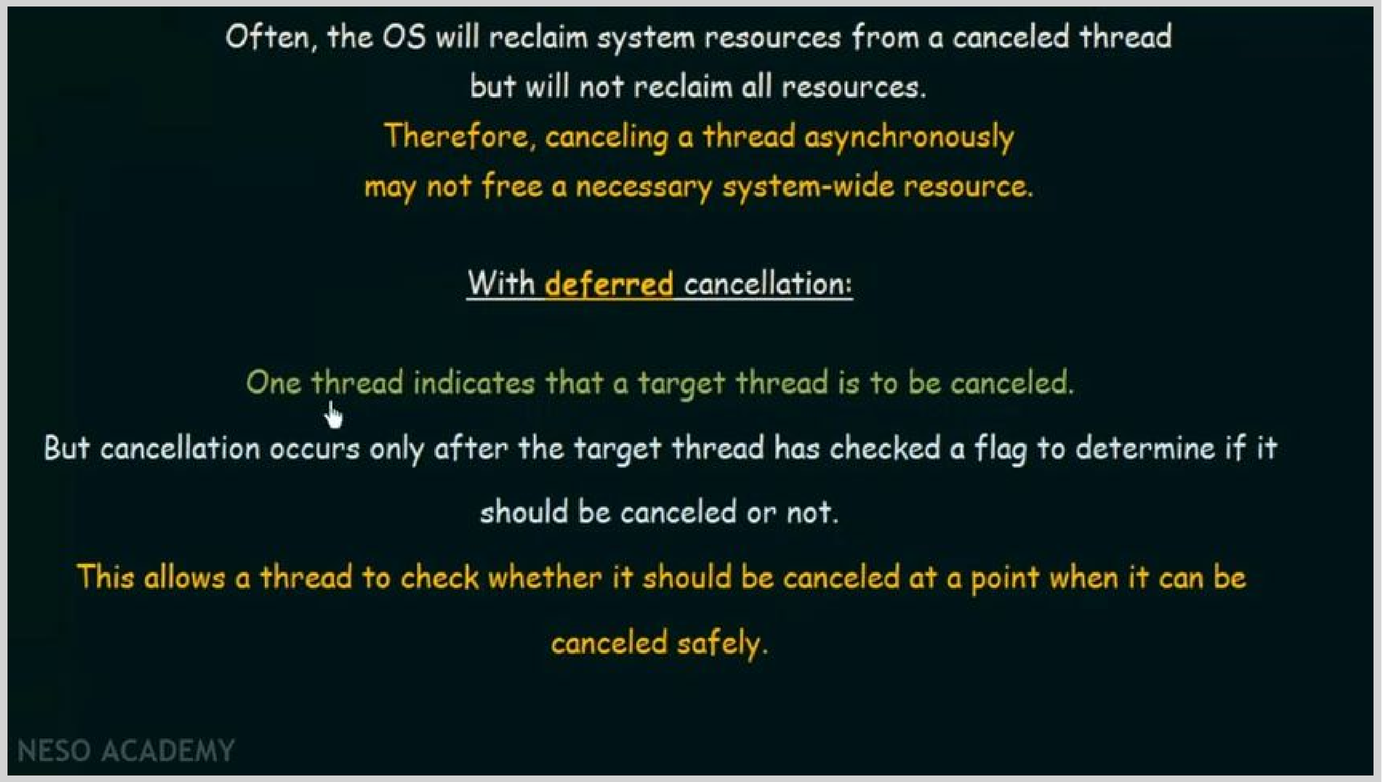
[08:33]


Where the difficulty with cancellation lies:
In situations where:
• Resources have been allocated to a canceled thread
• A thread is canceled while in the midst of updating data it is sharing with other threads.
Often, the OS will reclaim system resources from a canceled thread
but will not reclaim all resources.
Therefore, canceling a thread asynchronously
may not free a necessary system-wide resource.
NESO ACADEMY

PDF
PDF

[08:39]

←


Often, the OS will reclaim system resources from a canceled thread
but will not reclaim all resources.
Therefore, canceling a thread asynchronously
may not free a necessary system-wide resource.
With deferred cancellation:
One thread indicates that a target thread is to be canceled.
But cancellation occurs only after the target thread has checked a flag to determine if it should be canceled or not.
This allows a thread to check whether it should be canceled at a point when it can be canceled safely.
NESO ACADEMY

PDF
PDF

PDF

PDF