Sub: Operating System

Research topic: Threading Concepts in OS

Presented By :

556 :Vaishnavi Pawar

557: Yash Power

558: Devashish Pednekar

559: Dayasagar Phondekar

560: Aishwini Priya Pillai

Presented To:

Miss. Suvarna Kannav Mam

What is Thread?

- A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers..
- Threads within the same process share the same memory space and resources, making communication and data sharing between threads more efficient.

What is Multithreading?

- Multithreading is the ability of an application or process to execute multiple threads concurrently.
- Multithreading allows the application to perform multiple tasks simultaneously and efficiently utilize the available CPU resources.

Threads and Processes

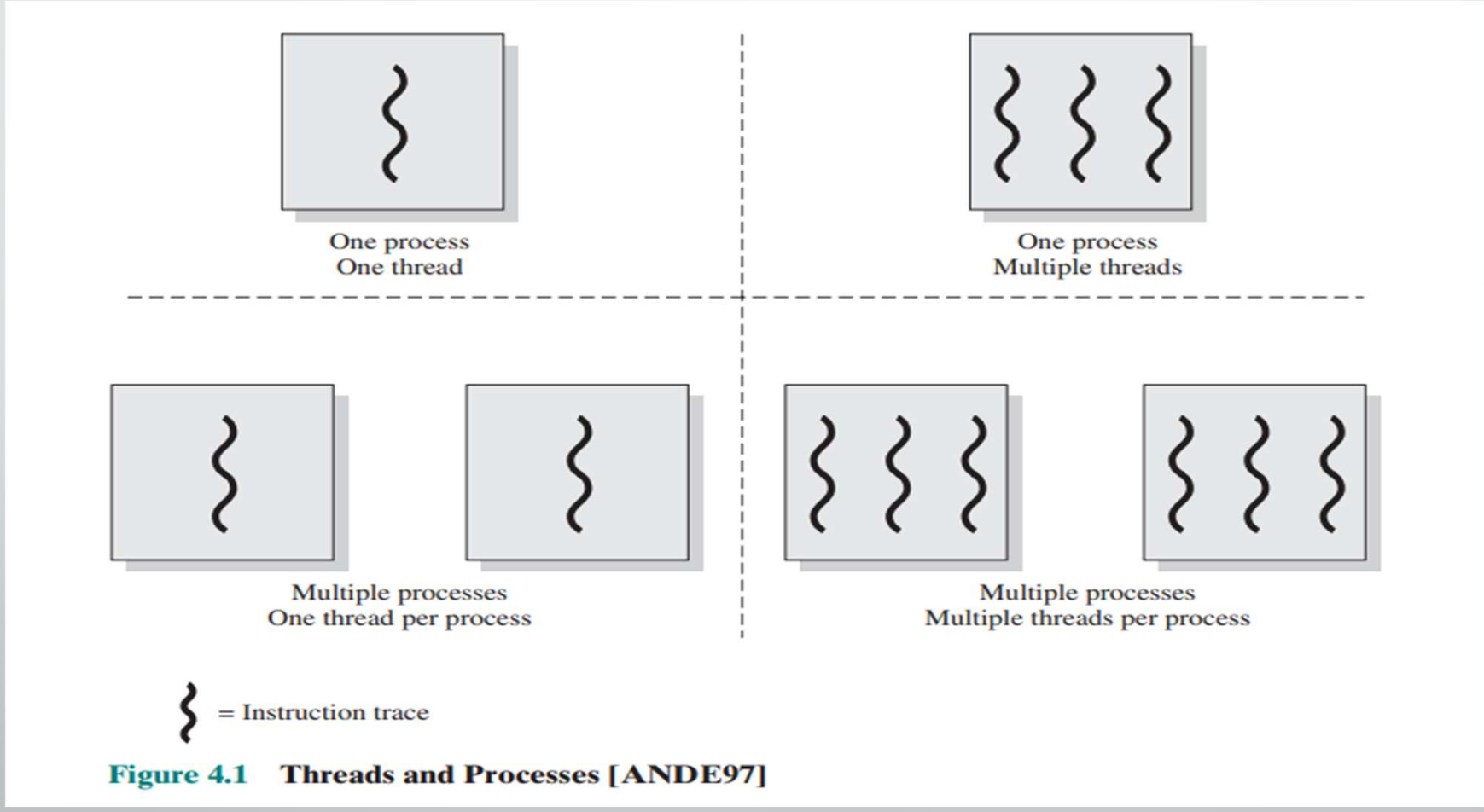

Figure 4.1 Threads and Processes [ANDE97]

Single – Threaded & Multithreaded process Models

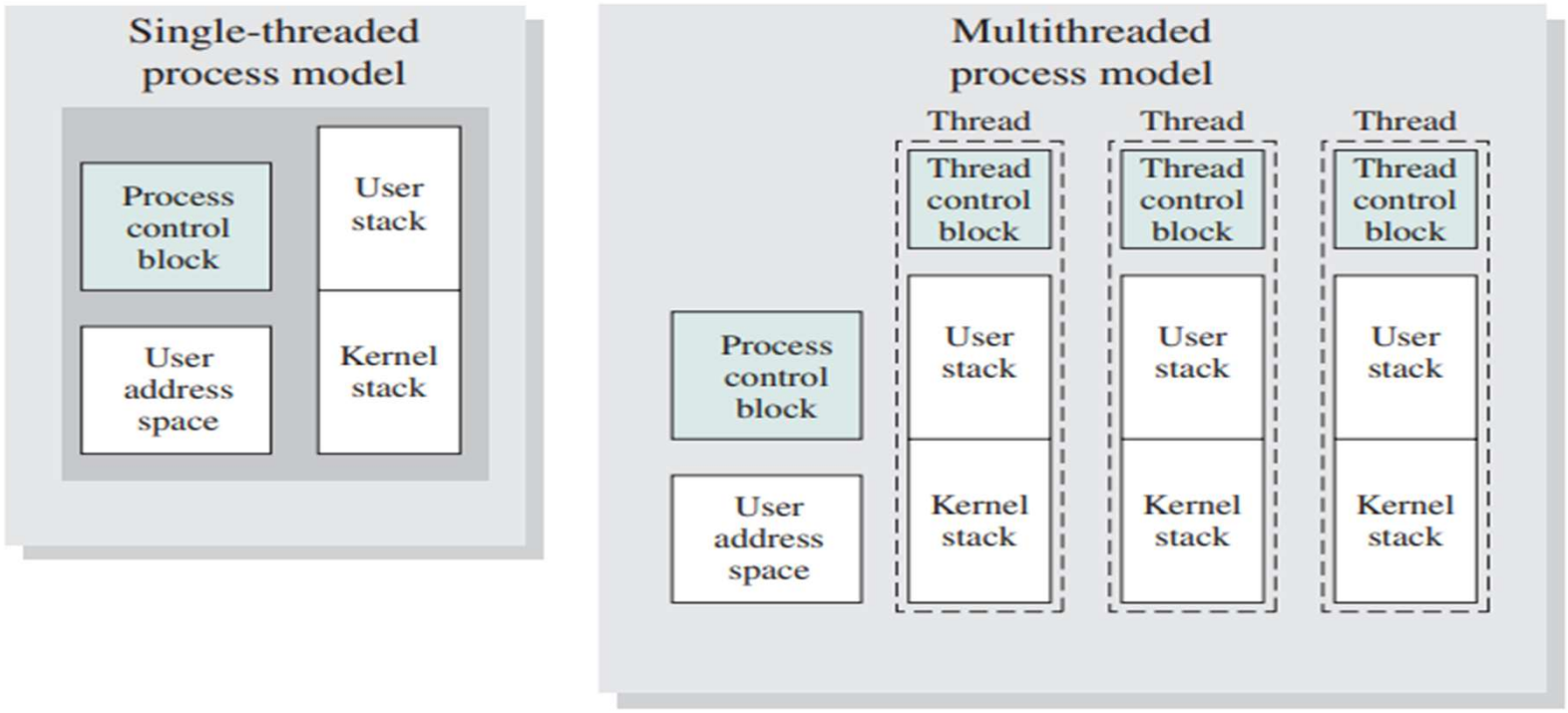

Figure 4.2 **Single-Threaded and Multithreaded Process Models**

Types of Threads :

1.User Level Thread .

2. Kernel-Level Threads .

3. Other Arrangements

1.User Level Threads:

- In a pure ULT facility, all of the work of thread management is done by the application and the kernel is not aware of the existence of threads.
- The thread management, creation, and synchronization are handled by a thread library implemented in user space.
- Since ULTs are independent of the operating system, they are more lightweight and have lower overhead when creating and switching between threads.
- User-level threads suffer from the limitation that if one thread blocks or enters into a system call, all threads within the same process are blocked, including threads running on different processor cores.

2. Kernel-Level Threads :

- In a pure KLT facility, all of the work of thread management is done by the kernel. There is no thread management code in the application level , Windows is an example of this approach.
- Each thread is represented as a separate kernel data structure and is scheduled and managed by the OS scheduler.
- Kernel-level threads provide better concurrency and parallelism since they can be scheduled independently, even if one thread is blocked.
- However, creating and managing kernel-level threads involve higher overhead due to system calls and kernel involvement.

3. Other Arrangements :

- The other two combinations have also been investigated, namely, a many-to-many relationship and a Many-to-One relationship.

1) Many-to-Many model:

- This model combines multiple user-level threads to a smaller or equal number of kernel-level threads.
- It allows for better control over thread management and can provide good concurrency while avoiding the limitations of the many-to-one model.
- The thread library handles the mapping between user-level and kernel-level threads.

2) Many-to-One model:

- This model uses many user-level threads mapped to a single kernel-level thread.
- The thread management is performed in user space, but when one thread blocks, the entire process blocks, as there is only one kernel thread handling all user-level threads.
- While lightweight, this model is limited in scalability and does not take full advantage of multi-core processors.

Thread Creation and Management:

- Threads can be created by the operating system or explicitly by the application using threading APIs provided by the OS or programming language.

- Thread management includes functions like thread creation, termination, joining and setting thread attributes.

- Thread States in Windows:

1. Ready:
    1. A ready thread is waiting to be scheduled for execution by the Kernel dispatcher.
    2. The Kernel dispatcher keeps track of all ready threads and schedules them in priority order.

2. Standby:
    1. A standby thread has been selected to run next on a specific processor.
    2. It waits in this state until the processor becomes available.
    3. If the standby thread's priority is high enough, it may preempt the currently running thread on that processor.
3. Running:
    1. Once the Kernel dispatcher switches to a thread for execution, it enters the Running state.
    2. The thread starts executing instructions on the CPU.

4.Waiting:

1. A thread enters the Waiting state when it is blocked on an event, such as I/O operation or synchronization purposes.
2. It may also voluntarily suspend itself.
3. The thread remains in this state until the waiting condition is satisfied.

5.Transition:

1. A thread enters the Transition state after waiting if it is ready to run, but the required resources are not available.
2. Once the necessary resources become available, the thread moves back to the Ready stat

6.Terminated:

1. A thread can be terminated either by itself, by another thread, or when its parent process terminates.

2.Alternatively, it may be retained by the Executive (kernel component) for future reinitialization.

Windows thread states

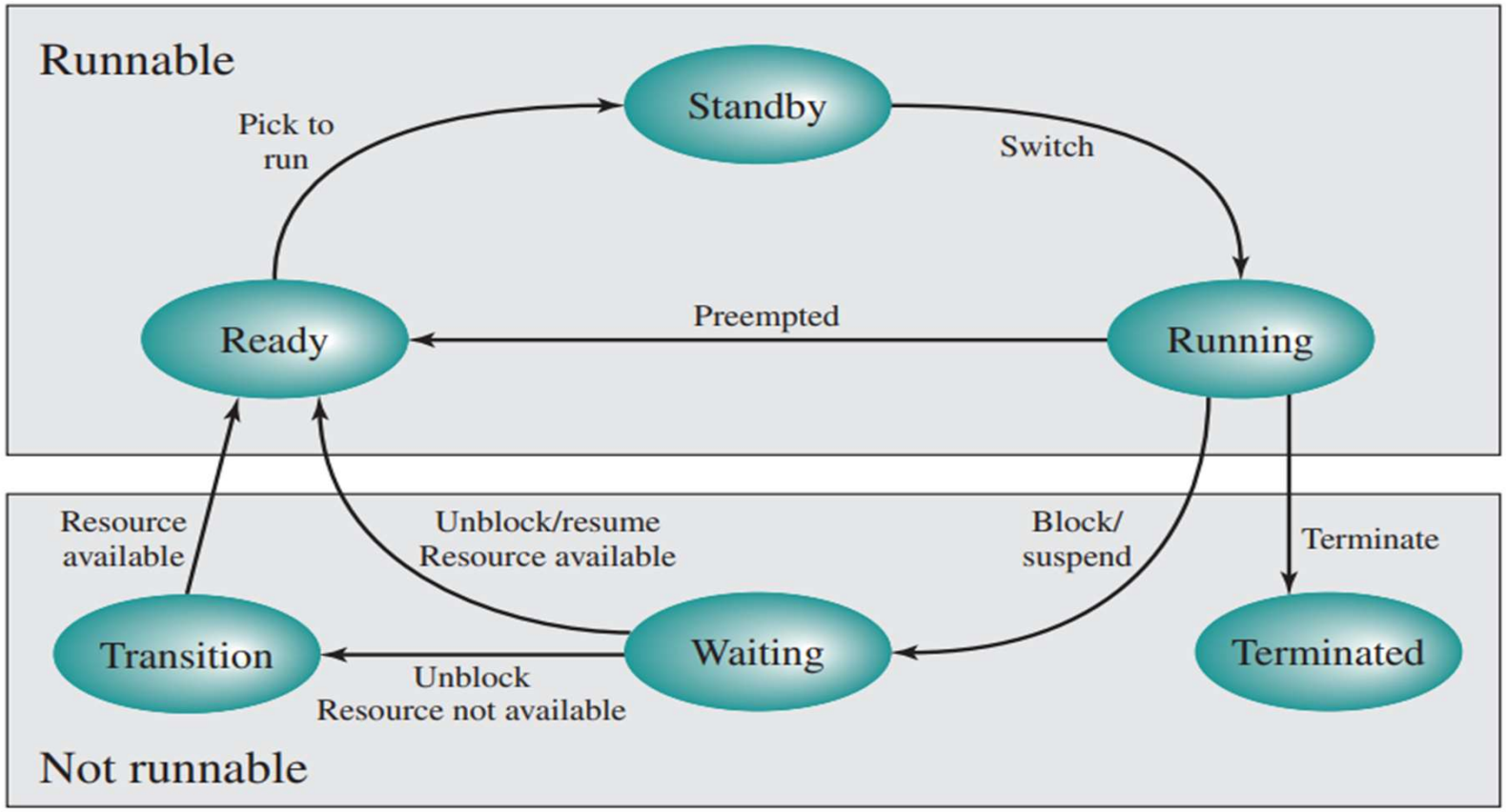

Figure 4.12 **Windows Thread States**


Thank You!
CAREER PROTOCOL