Module 2.2 Thread

Swati Mali

Nirmala Shinde Baloorkar

Assistant Professor

Department of Computer Engineering

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya
TRUST

Outline

- Thread
- Process Vs Thread
- Example

Thread

- Supports Parallelism with multiple threads of execution at a time
- A thread executes sequentially and is interrupt-able so that the processor can turn to another thread
- Does not need entire process context to execute so considered as lightweight.
- Includes the program counter and stack pointer) and its own data area for a stack
- Supports multiple parallel executions

  e.g. Notifications in background while you are using the app
- The idea is to **achieve parallelism** by **dividing a process** into **multiple threads.**
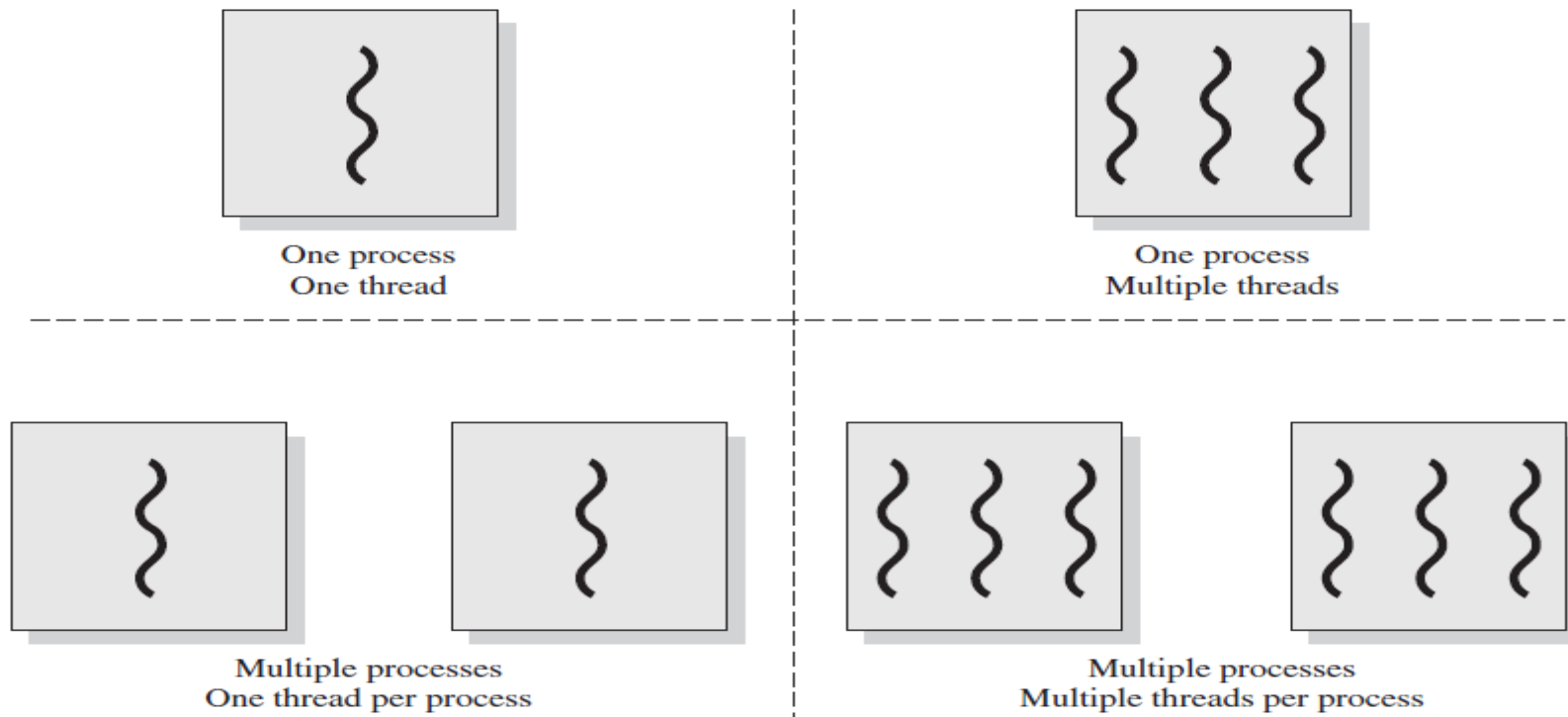
Process and Thread

- A process is an instance of a program in execution.

- It is a basic unit of work that can be scheduled and executed by the operating system.

- A thread is the unit of execution within a process.

- A process can have anywhere from just one thread to many threads.
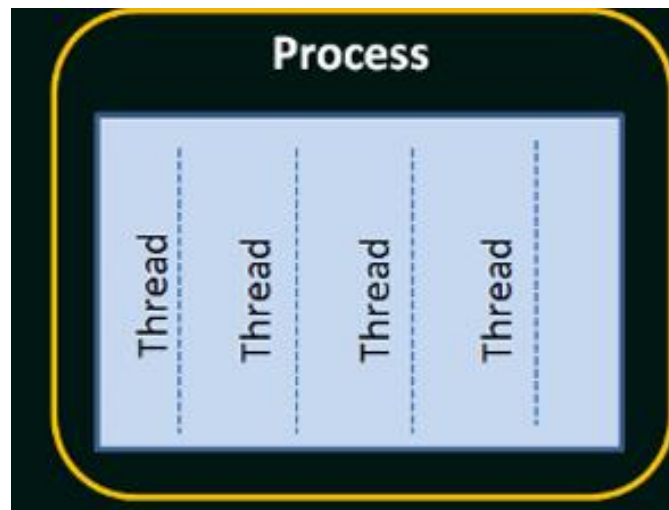
Thread

- *Multithreading* refers to the ability of an OS to support multiple, concurrent paths of execution within a single process.

One process
One thread

One process
Multiple threads

Multiple processes
One thread per process

Multiple processes
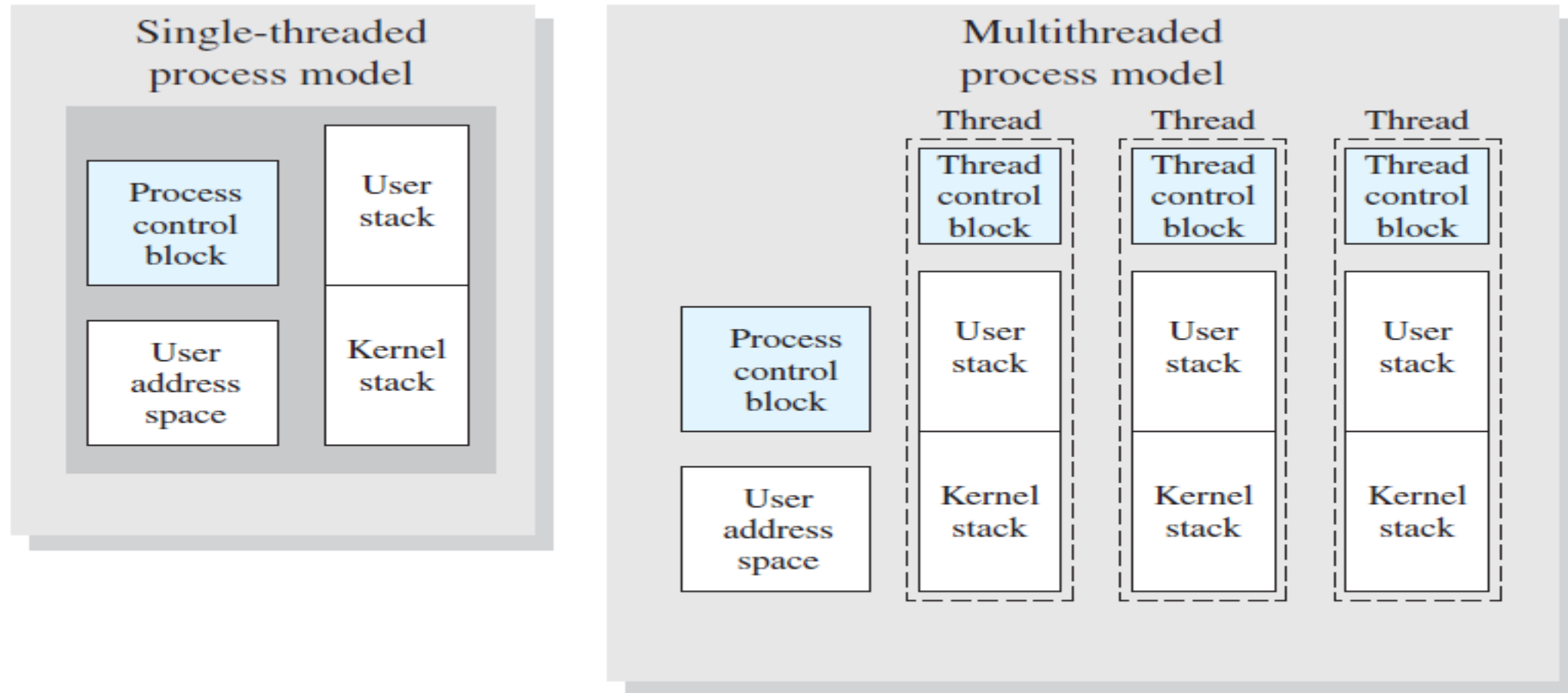Multiple threads per process

Thread

Figure 4.2   Single Threaded and Multithreaded Process Models

The key benefits of threads

- It takes far less time to create a new thread in an existing process than to create a brand-new process.

- It takes less time to terminate a thread than a process.

Types of threads

- Kernel level threads : managed by kernel
- User level threads : managed by user with support from programming languages and libraries
- Hybrid threads

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya
TRUST

Thread

- Thread libraries provide programmers with API for the creation and management of threads.
- Three types of Thread
  - **POSIX Pitheads** may be provided as either a user or kernel library, as an extension to the POSIX standard.
  - **Win32 threads** are provided as a kernel-level library on Windows systems.
  - **Java threads**: Since Java generally runs on a Java Virtual Machine, the implementation of threads is based upon whatever OS and hardware the JVM is running on, i.e. either Pitheads or Win32 threads depending on the system.

Thread

- Create
- Join
- Terminate

Create, Start and Join

- A thread can be created using the **Thread class** provided by the threading module. Using this class, you can create an instance of the Thread and then start it using the **.start()** method.

Create & Start

```python
import threading

# Creating Target Function

def num_gen(num):
    for n in range(num):
        print("Thread: ", n)


# Main Code of the Program

if __name__ == "__main__":
    print("Statement: Creating and Starting a Thread.")
    thread = threading.Thread(target=num_gen, args=(3,))
    thread.start()
    print("Statement: Thread Execution Finished.")
```

- 1st execution

```
Statement: Creating and Starting a Thread.
Thread:  0
Statement: Thread Execution Finished.
Thread:  1
Thread:  2
```

- 2nd execution

```
Statement: Creating and Starting a Thread.
Thread:  0
Thread:  1
Statement: Thread Execution Finished.
Thread:  2
```

Join() Method

- The join() method is used in that situation, it doesn't let execute the code further until the current thread terminates.

```python
import threading

# Creating Target Function
def num_gen(num):
    for n in range(num):
        print("Thread: ", n)


# Main Code of the Program
if __name__ == "__main__":
    print("Statement: Creating and Starting a Thread.")
    thread = threading.Thread(target=num_gen, args=(3,))
    thread.start()
    thread.join()
    print("Statement: Thread Execution Finished.")
```

```
Statement: Creating and Starting a Thread.
Thread:  0
Thread:  1
Thread:  2
Statement: Thread Execution Finished.
```

Assignment

- ULT vs KLT
- Pthread
- POSIX

Question ?