Thread ch4

By

Lecturer: Ameen A.Noor

Thread

- A thread is a stream of instructions (line of control) within a process that can be executed independently of other threads.
- This means that a process may create at sometimes several threads that can be executed concurrently by several processors or each thread is dispatched for one time slice .
- Another definition of thread is a "Light Weight Process LWP" as it simulates the original process "Called Heavy Weight Process HWP" in the running for one time slice when it is dispatched.
- As the thread runs in a process environment, therefore, it shares a process address space which means that communication between threads is very simple and variable sharing is possible without causing address violation problem.

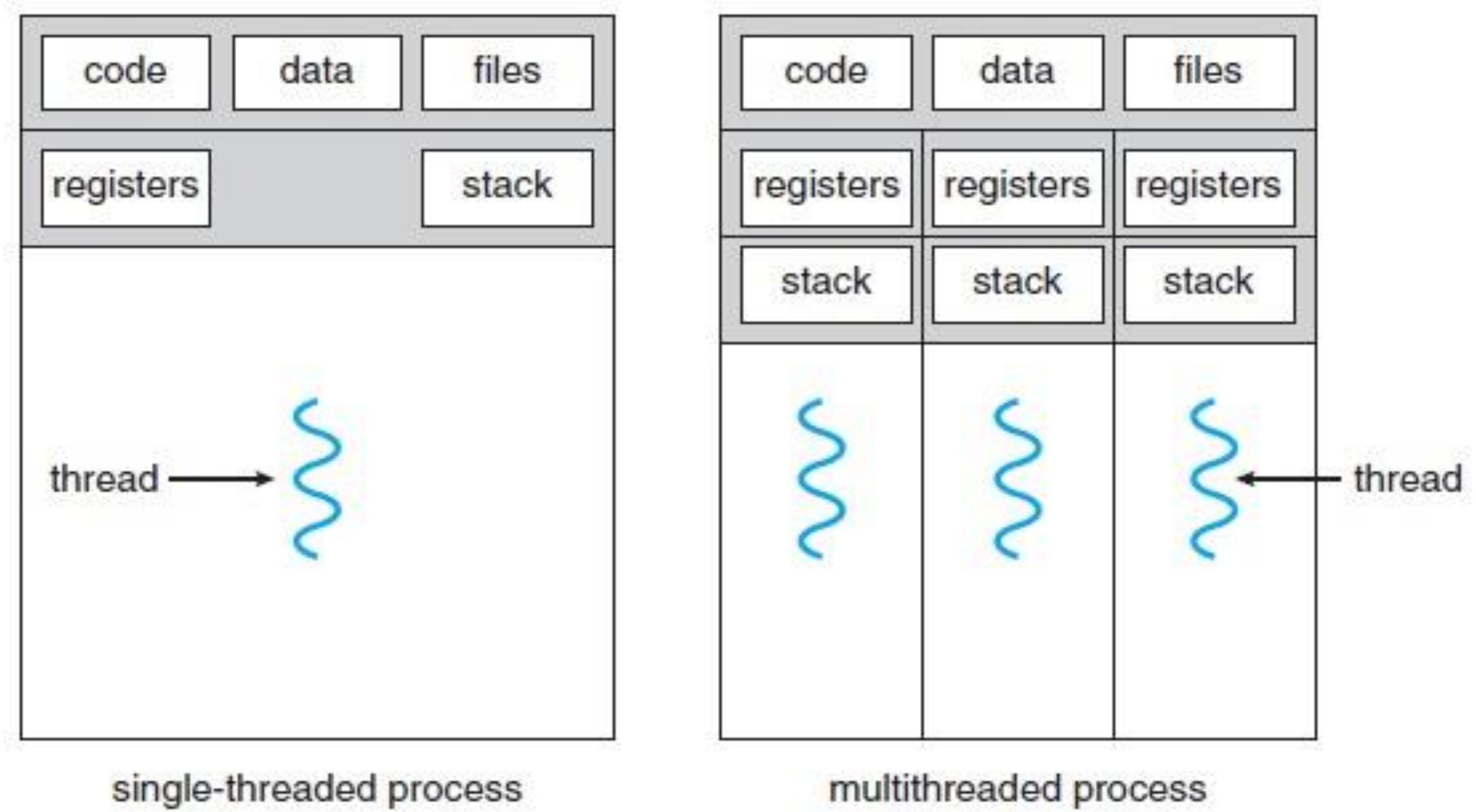
Motivations of Threads

1. Fast execution of a program as it can make use of several processors at the same time (case of multiprocessing) or dispatched more time slices (Case of Single Processor CPU)
2. Easy communication between threads as they share the same process address space that created them.
3. Easier design of some applications which have a lot of parallel activities such as a "Word" program.

There are two types of programming languages

1. Single threaded: Allows single thread of control in the program and hence concurrent activities are not possible within the same program. Examples of such language are: C, C++, VB, etc.
2. Multithreaded: Allows several threads of control in the program and hence concurrent activities are quite possible within the same program. Examples of such languages are: C#, VB.NET, JAVA, ADA, etc.

- It is worth noting that "single threaded languages" are also called "non-threaded" or "sequential" while "multithreaded" are called "threaded" or "parallel".



Non-Threaded and Threaded Algorithms

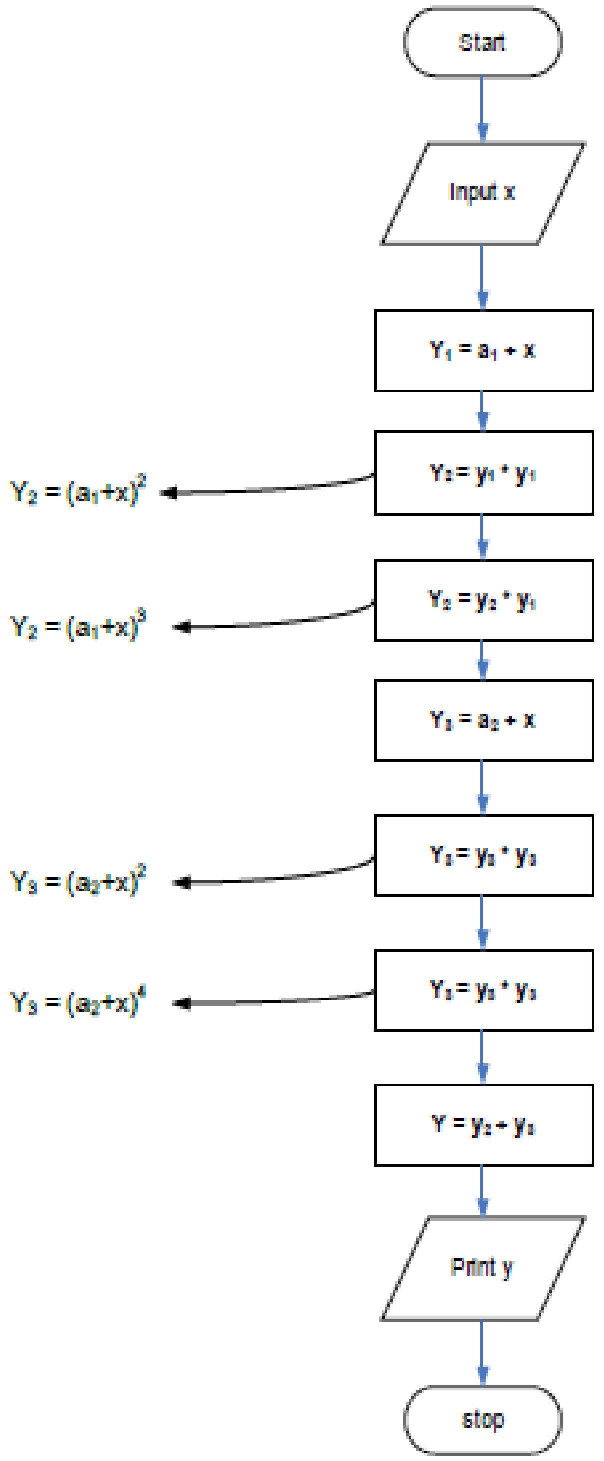
- Suppose we want to calculate the following expressions:

$$Y= (a_1+x)^3 + (a_2+x)^4$$

where a_1, a_2 are constants and x is input variable. This calculation can be done as follows:

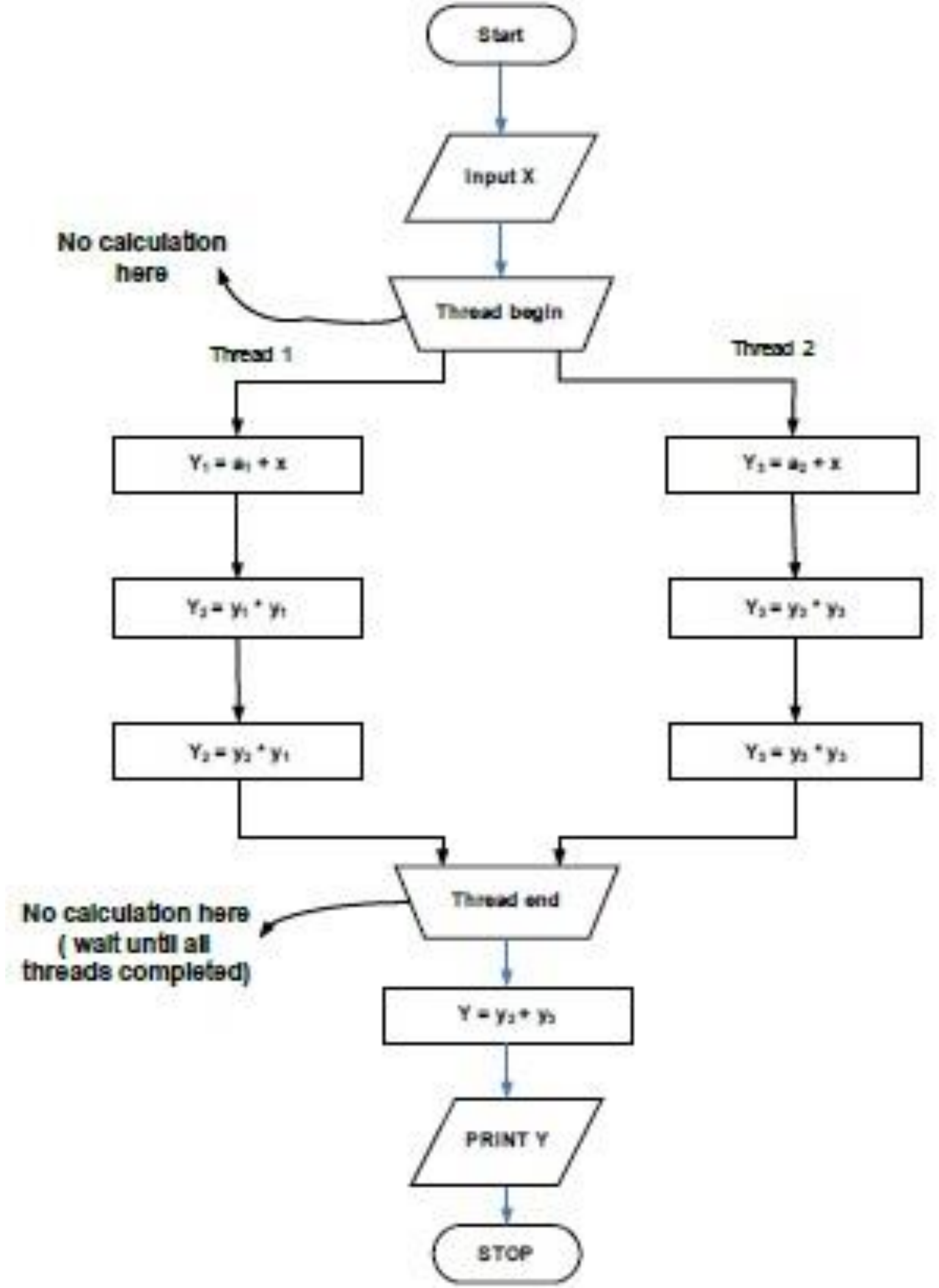
1. **Using Non-Threaded Algorithm:**

   The calculation is shown in figure bellow and we notice that it takes a total of 7 arithmetic operations


Start
Input x
$Y_1 = a_1 + x$
$Y_2 = y_1 * y_1$
$Y_2 = (a_1+x)^2$
$Y_2 = y_2 * y_1$
$Y_2 = (a_1+x)^3$
$Y_3 = a_2 + x$
$Y_3 = y_3 * y_3$
$Y_3 = (a_2+x)^2$
$Y_3 = y_3 * y_3$
$Y_3 = (a_2+x)^4$
$Y = y_2 + y_3$
Print y
stop

2. **Using Threaded Algorithm:**

The calculation is shown in fig below the number of arithmetic operations in Thread1 is 3, and in Thread2 is 3.



Threading Models

1- User-Level Threads

- The first method is to put the threads package entirely in user space, the kernel knows nothing about them, so User level threads perform threading operations in the user space, meaning that threads are created by runtime libraries that cannot execute privileged instructions or access kernel primitives directly.

2- Kernel-Level Threads

Kernel-level threads attempt to address the limitations of user-level threads by mapping each thread to its own execution context.

3- Combining User- level Threads and kernel-level Threads

Also called hybrid Threads, various ways have been investigated to try to combine the advantages of user-level threads with kernel-level threads. One way is use kernel-level threads and then multiplex user-level threads onto some or all of the kernel threads