PART 5

THREADS LIBRARIES

Dr. Soha S. Zaghloul

LAYOUT

- Threading Modes
  - Asynchronous Threading
  - Synchronous Threading
- Thread Libraries
  - POSIX & Pthreads
  - Windows Threads Library
  - Java Threads Library
- Threads Manipulation
  - Java

QUICK REVIEW

A thread library provides the programmer with an Application Programming Interface (API) for creating and managing threads.

There are two primary ways for the implementation of a thread library:

- User-level: the library (code & data structures) is totally provided in the user space.
  - There is no kernel support.
  - Calling a function from the library is actually a local function call rather than a system call.
- Kernel-level: the library (code & data structures) is provided in the kernel space.
  - The library is directly supported by the OS.
  - Calling a function from the library results in a system call.

THREADING MODES (1) – INTRODUCTION

There are two modes of threading:

- Synchronous threading
- Asynchronous threading

THREADING MODES (2) – ASYNCHRONOUS THREADING

Once the parent spawns a child, asynchronous threading is characterized by the following:

- Both threads run concurrently.
- Each thread runs independently of each other.
- The parent needs to know when its child terminates.
- Typically, there is little data sharing between parent and child.

THREADING MODES (3) – SYNCHRONOUS THREADING

Once the parent spawns a child, synchronous threading is characterized by the following:

- The parent waits for the termination of all its children before resuming execution: this is called the fork-join strategy.
- All children run concurrently.
- Once a child thread finishes its job, it terminates and joins the parent.
- Typically, it involves significant data sharing among threads. For example, the parent thread may combine the results calculated by its various children.

THREADS LIBRARIES – INTRODUCTION

Three main thread libraries are currently used by programmers:

- POSIX Pthreads
- Windows threads
- Java threads

THREADS LIBRARIES – POSIX & PTHREADS

POSIX is a set of standards set by IEEE and specified as IEEE1003.1c.

POSIX standards define rules for thread creation and synchronization.

The details implementation is up to the library development.

POSIX is commonly used in UNIX-based OSs (Linux, Solaris MAC OS X).

Pthreads is an extension to POSIX.

Pthreads may be provided on both levels: user-level and kernel-level.

Pthreads apply synchronous threading: fork-and-join.

Data in threads may be identified as:

- Global data:
  - These are shared among all threads belonging to the same process.
- Thread-local data:
  - These are declared locally in a function.
  - Therefore, they are stored in the stack.
  - Since each thread has its own stack, then each thread has its own copy of local data.

THREADS LIBRARIES – WINDOWS

- Implemented on Windows OS.
- Implemented as kernel-level library.
- Global data are shared between threads.
- Each thread may have its own copy of local data.
- Windows applies synchronous threading mode

THREADS LIBRARIES – JAVA

Java thread API allows programmers to create and manage threads directly in Java programs.

Implemented as user-level library.

Since JVM runs on top of an OS, then the Java thread API itself is generally implemented using the library on the host system.

For example, on Windows systems, Java threads are typically implemented using the Windows API.

Java has no notion of global data. Any variable belongs to either a method or a class. Therefore, access to shared data must be explicitly arranged between threads.

On the other hand, each thread has its own copy of locally declared data.

THREADS MANIPULATION – JAVA (1)

The following code is an example of a program that checks if the arguments are odd or even,

Let us write first the sequential program:

```
1   class Sum
2   {
3     private int sum = 0;
4     public int getSum() {
5       return sum;
6     } /*end getSum */
7     public void setSum (int sum) {
8       this.sum += sum;
9     } /* end of setSum */
10  } /* end of class Sum */
11
12  class Summation implements Runnable
13  {
14    private int upper;
15    private Sum sumValue;
16    public Summation (int upper, Sum sumValue) {
17      this.upper = upper;
18      this.sumValue = sumValue;
19    } /*end of Summation constructor */
20    public void run() {
21      int sum = 0;
22      for (int i= 0; I <= upper; i++)
23         sum += i;
24       sumValue.setSum (sum);
25    } /* end run() */
26  } /* end class Summation */
```

THREADS MANIPULATION – JAVA (I)

The following program is an example of multithreading programming using Java.

The program calculates the summation of non-negative numbers.

```
1   class Sum /* this is the variable shared between all threads */
2   {
3     private int sum = 0;
4     public int getSum() {
5       return sum;
6     } /*end getSum */
7     public void setSum (int sum) {
8       this.sum += sum;
9     } /* end of setSum */
10  } /* end of class Sum */
11
12  class Summation implements Runnable /*this is the interface*/
13  {
14    private int upper;
15    private Sum sumValue;
16    public Summation (int upper, Sum sumValue) {
17      this.upper = upper;
18      this.sumValue = sumValue;
19    } /*end of Summation constructor */
20    public void run() { /* this is the code implemented by each thread independently using local-thread variables */
21      int sum = 0;
22      for (int i= 0; I <= upper; i++)
23          sum += i;
24       sumValue.setSum (sum);
25    } /* end run() */
26  } /* end class Summation */
27
```

THREADS MANIPULATION – JAVA (2)

```
28  public class Driver
29  {
30    public static void main (String[] args) { /*threads are manipulated (created, started, synchronized, etc..) here*/
31      if (args.length > 0)
32        if (Integer.parseInt(args[i]) < 0)
33          System.err.println (args[i]  + "must be >= 0");
34        else {
35          Sum sumObject = new Sum();
36          int upper = Integer.parseInt(args[1]);
37          Thread thrd = new Thread(new Summation(upper, sumObject));
38          thrd.start();
39          try {
40            thrd.join();
41            System.out.println ("The sum of " + upper + " is " + sumObject.getSum());
42          } catch (InterruptedException ie) { }
43        } /* end else */
44     } /* end if (args.length > 0) */
45    else
46      System.err.println ("Usage: Summation <integer value>");
47   } /* end main */
48  } /* end class Driver */
```

THREADS MANIPULATION – JAVA (3)

```
1   class Sum
2   {
3     private int sum =0;
4     public int getSum() {
5       return sum;
6     } /*end getSum */
7     public void setSum (int sum) {
8       this.sum += sum;
9     } /* end of setSum */
10  } /* end of class Sum */
```

Since there is no global data in Java, the class Sum is defined for the variables to be shared.

This class is accessible by all threads: Refer to the Summation & the Driver classes.

THREADS MANIPULATION – JAVA (4)

Java threads are typically implemented using the thread model provided by the underlying OS.

There are two techniques to create threads in Java:

- Using Java Runnable Interface
- Overriding the run() method

THREADS MANIPULATION – JAVA (5) – RUNNABLE INTERFACE

The following steps are followed:

1) Declare a class C that implements the Runnable interface.

2) C implements the run() method.

3) An instance of C is then allocated and passed as a parameter when creating a thread.

Consider the following example that finds the prime numbers greater than a specified value:

```
1  class PrimeRun implements Runnable {
2    long minPrime;
3    PrimeRun (long minPrime) {
4      this.minPrime = minPrime;
5    } /* end of constructor */
6    public void run() {
7      //compute the primes larger than minPrime
8      …
9    } /* end run() */
```

Then in main(), create a thread and start it running:

```
m  PrimeRun p = new PrimeThread (1430003333133222);
n  new Thread(p).start();
```

This approach is more commonly used than the second one.

THREADS MANIPULATION – JAVA (6) – OVERRIDING

The following steps are followed:

1) Declare a class *C* as a subclass of the Thread class

2) *C* should override the run() method of the Thread class.

3) An instance of *C* is then allocated and started.

Consider the following example that finds the prime numbers greater than a specified value:

```
1   class PrimeThread extends Thread {
2     long minPrime;
3     PrimeThread (long minPrime) {
4       this.minPrime = minPrime;
5     } /* end of PrimeThread constructor */
6
7     public void run() {
8       // computes the prime numbers larger than minprime
9       …
10    } /* end run() */
```

Then in main(), create a thread and start it running:

```
m   PrimeRun p = new PrimeThread (143);
n   p.start();
```

THREADS MANIPULATION – JAVA (7)

```
12  class Summation implements Runnable
13  {
14    private int upper;
15    private Sum sumValue;
16    public Summation (int upper, Sum sumValue) {
17      this.upper = upper;
18      this.sumValue = sumValue;
19    } /*end of Summation constructor */
20    public void run() {
21      int sum = 0;
22      for (int i= 0; i <= upper; i++)
23          sum += i;
24       sumValue.setSum (sum);
25    } /* end run() */
26  } /* end class Summation */
```

The Summation class Implements the Runnable interface. *(line 12)*

The Summation constructor *(lines 16 to 19)*

The run() method implements the code to be executed by each thread *(lines 20 to 25)*

THREADS MANIPULATION – JAVA (8)

37	**Thread thrd = new Thread(new Summation(upper, sumObject));**
	new Summation(upper, sumObject) instantiates an object from the class Summation that implements the Runnable interface. Let it be p.
	Thread thrd = new Thread(p) ➔ instantiates a new thread (thrd) from the Thread class.

38	**thrd.start();**
	Start running the thread thrd acccording to the code written in the run() method.
	Note that we never call the run() method directly.

40	**thrd.join();**
	Waits for the completion of thrd. The thread then joins the parent process.
	thrd.join() can throw an InterruptedException. However, it is ignored in this code.