Nội dung môn học

Chương 1: Giới thiệu về NNLT Java

Chương 2: Hướng đối tượng trong Java

Chương 3: Các lớp tiện ích trong Java

Chương 4: Quản lý Exception

Chương 5: Nhập xuất dữ liệu (I/O Streams)

Chương 6: Xử lý đa luồng

Chương 7: Kết nối và thao tác CSDL với JDBC

Chương 8: Lập trình GUI với AWT & Swing

ĐA LUỒNG TRONG JAVA

(JAVA MULTITHREADING)

Nội dung

1. Giới thiệu về process và thread.
2. Vòng đời thread.
3. Độ ưu tiên thread.
4. Cách tạo thread.
5. Đồng bộ hóa các thread
6. Ví dụ minh họa
   - Multi Cores
   - Multi-Client và Server

Giới thiệu về process và thread

Hệ điều hành đa nhiệm cổ điển

- Đơn vị cơ bản sử dụng CPU là process.
- Mỗi process thi hành 1 ứng dụng riêng.
- Mỗi process có không gian địa chỉ và không gian trạng thái riêng.

Giới thiệu về process và thread.

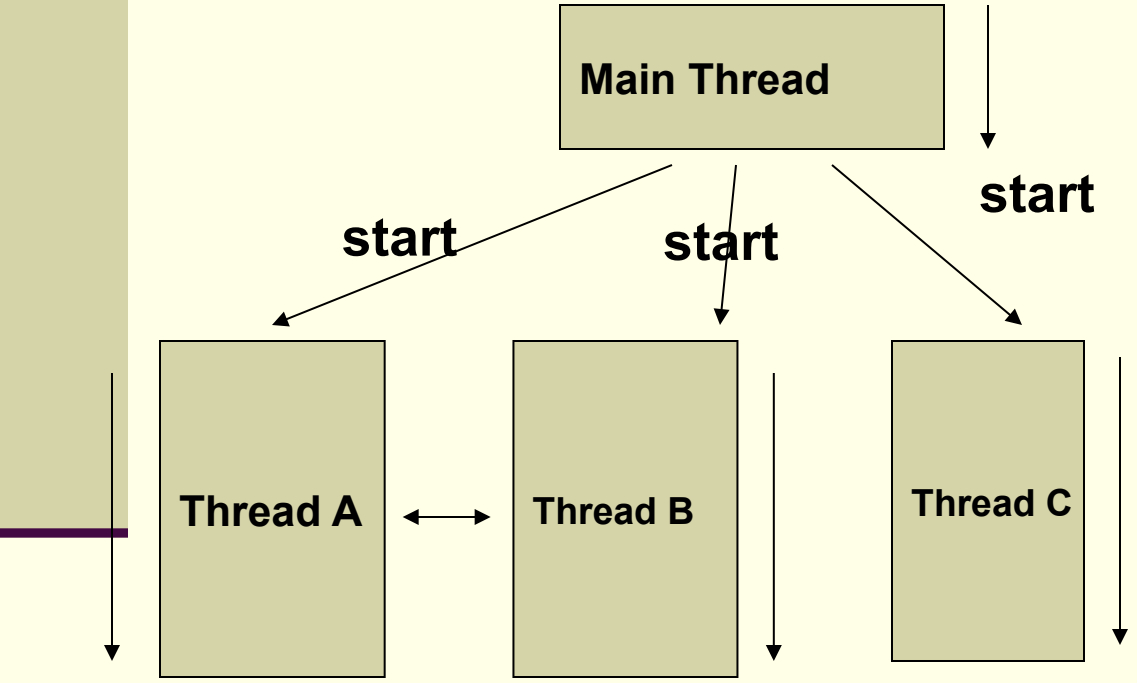
Hệ điều hành đa nhiệm hiện đại:

- Đơn vị cơ bản sử dụng CPU là thread
- Mỗi process có không gian địa chỉ và nhiều thread. Mỗi thread có bộ đếm chương trình, trạng thái các thanh ghi và ngăn xếp riêng.
- Các thread của một process có thể chia sẻ nhau không gian địa chỉ, biến toàn cục, tập tin, chương trình con.
- Thread cung cấp cơ chế tính toán song song cho ứng dụng.

Giới thiệu về process và thread

- 2 đơn vị thực thi cơ bản trong lập trình đồng thời: **Process &Thread**.
- **Multitasking là nhiều process** chia sẻ tài nguyên dùng chung như CPU. **Multithreading mở rộng ý tưởng multitasking vào ứng dụng**, phân chia nhiều công việc khác nhau trong ứng dụng thành các thread.
- Thời gian xử lý cho 1 core CPU đơn được chia sẻ giữa các process và các thread thông qua HĐH.
- Lập trình đồng thời trong Java → tập trung vào thread.

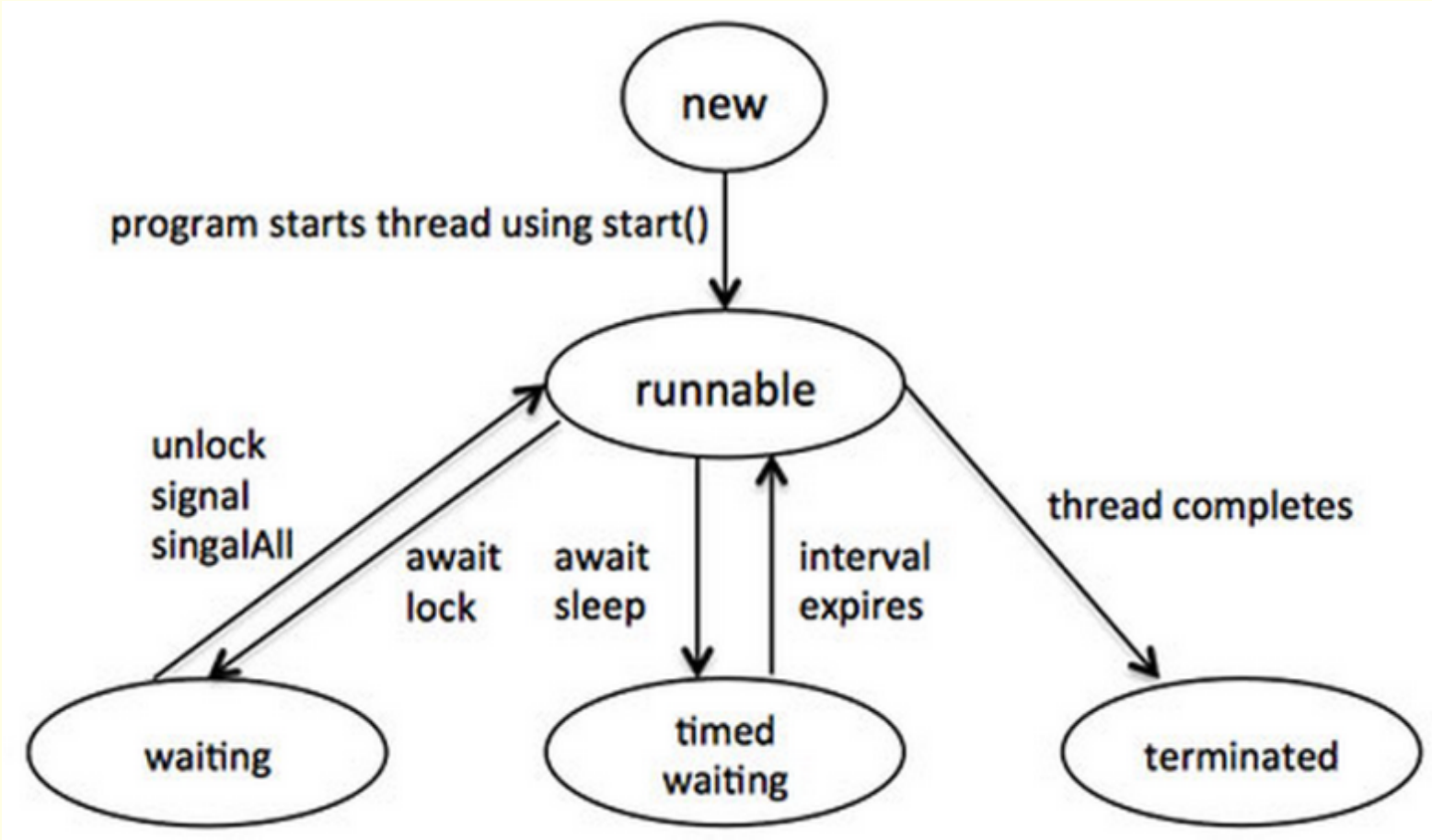
Giới thiệu về process và thread



Các thread có thể chuyển đổi dữ liệu với nhau

- Mỗi phần thực hiện các công việc khác nhau cùng thời điểm → **tối ưu sử dụng tài nguyên (CPUs)**.
- OS không chỉ phân chia thời gian xử lý giữa các ứng dụng, mà còn giữa các thread trong một ứng dụng.

Vòng đời của một thread



Vòng đời của một thread

- **New**: bắt đầu vòng đời một thread được tạo ra.
- **Runnable**: sau khi thread khởi động dùng start(). Thread đang thực thi công việc của nó.
- **Waiting**: chuyển sang trạng thái **waiting** khi thread đợi thread khác thực hiện một công việc. Chuyển về **runnable** khi thread khác thông báo cho thread đợi tiếp tục thực thi.
- **Timed waiting**: thread đang chạy chuyển sang trạng thái **time waiting** trong một khoảng thời gian xác định và chuyển về **runnable** khi hết thời gian đợi.
- **Terminated**: khi hoàn thành công việc hoặc các kết thúc khác.

Độ ưu tiên

- Mỗi thread có độ ưu tiên giúp OS xác định thứ tự mà các thread được lập lịch để thực thi.
- Độ ưu tiên của Java Thread
  - MIN_PRIORITY (1)
  - NORM_PRIORITY (5)
  - MAX_PRIORITY (10)
- Thông thường, thread có độ ưu tiên cao trong chương trình nên được cấp phát thời gian CPU trước các thread có độ ưu tiên thấp hơn.

Tạo thread dùng Runnable interface

- **Bước 1**: Hiện thực phương thức **run()** trong **Runnable** interface. Đây là entry-point cho thread và chúng ta nên đặt tất cả business logic của thread trong phương thức này.

  public void run( )

- **Bước 2**: Khởi tạo đối tượng Thread dùng constructor

  Thread(Runnable threadObj, String threadName);

- **Bước 3**: Khởi động Thread bằng cách gọi phương thức **start()**. Phương thức này sẽ gọi phương thức **run()**.

  void start( );

```
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;

   RunnableDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}
```

Tạo thread dùng Runnable interface

```
public class TestThread {
   public static void main(String args[]) {

      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Tạo thread dùng lớp Thread

- **Bước 1**: Overide phương thức **run()** trong lớp Thread. Phương thức này cung cấp entry-point cho thread.

  public void run( )

- **Bước 2**: Khởi động thread bằng cách gọi phương thức **start()**. Phương thức này sẽ gọi phương thức ***run()***.

  void start( );

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}
```

Tạo thread dùng lớp Thread

```
public class TestThread {
   public static void main(String args[]) {

      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Đồng bộ thread

- Khi hai hay nhiều thread trong 1 chương trình cùng cập một tài nguyên cùng lúc và cho ra kết quả bất ngờ vì vấn đề đồng thời.
- Nhiều thread cùng ghi dữ liệu vào 1 file → có thể gây hỏng dữ liệu, vì một trong những thread khác có thể ghi đè.

→ Cần đồng bộ hóa hành động của các thread

→ Đảm bảo chỉ một thread có thể truy cập tài nguyên ở một thời điểm.

```
synchronized(objectidentifier) {
   // Access shared variables and other shared resources
}
```

Ví dụ không đồng bộ

```
class PrintDemo {
   public void printCount(){
    try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Counter   ---    "  + i );
         }
     } catch (Exception e) {
         System.out.println("Thread  interrupted.");
     }
   }

}
```

Ví dụ không đồng bộ

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;

   ThreadDemo( String name,  PrintDemo pd){
       threadName = name;
        PD = pd;
   }
   public void run() {
     PD.printCount();
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}
```

Ví dụ không đồng bộ

```
public class TestThread {
   public static void main(String args[]) {

      PrintDemo PD = new PrintDemo();

      ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
      ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );

      T1.start();
      T2.start();

      // wait for threads to end
      try {
         T1.join();
         T2.join();
      } catch( Exception e) {
         System.out.println("Interrupted");
      }
   }
}
```

Ví dụ không đồng bộ

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   5
Counter   ---   2
Counter   ---   1
Counter   ---   4
Thread Thread - 1  exiting.
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Ví dụ có đồng bộ

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;

   ThreadDemo( String name,  PrintDemo pd){
       threadName = name;
       PD = pd;
   }
   public void run() {
     synchronized(PD) {
        PD.printCount();
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}
```

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 1  exiting.
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Multi Cores + Multi Threads

- Xử lý song song → Tận dụng tối đa tài nguyên CPU
- Các bước thực hiện:
  - Kiểm tra số CPU có thể dùng & Tạo ThreadPool với số CPU có thể.
  - Lặp: Khởi tạo 1 Thread mới cho mỗi công việc cần song song.

Multi Cores + Multi Threads

- *Kiểm tra số CPU có thể dùng & Tạo ThreadPool với số CPU có thể.*

```
Runtime runtime = Runtime.getRuntime();

int numOfProcessors = runtime.availableProcessors();

ExecutorService executor =

Executors.newFixedThreadPool(numOfProcessors - 1);
```

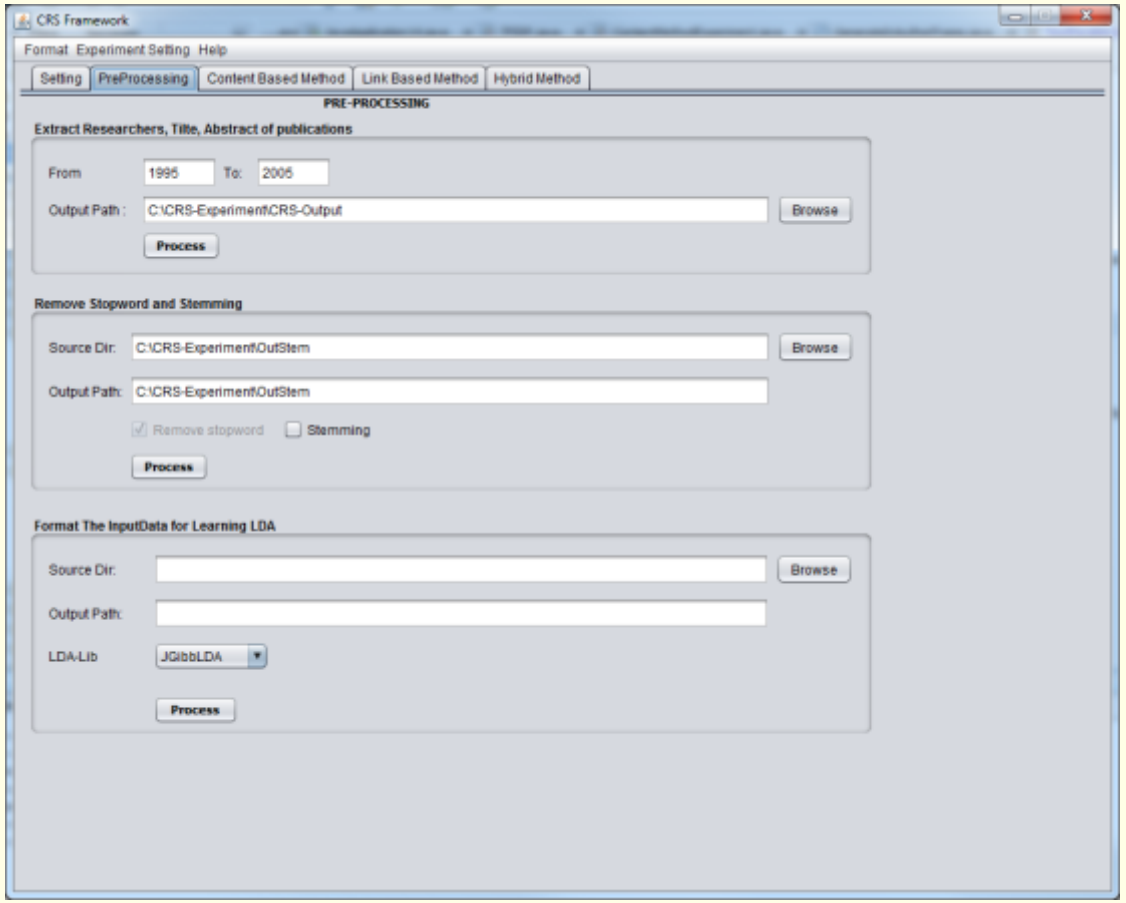
Multi Cores + Multi Threads

- Lặp: Khởi tạo 1 Thread mới cho mỗi công việc cần song song.

```
For (int i = 1; i<= NumberOfTasks; i++) {
    executor.submit(new Runnable() {
        @Override
        public void run() {
            // Tạo Thread & xử lý Task thứ i.
            call_function_to_process(Tasks[i]);
        }
    });
}
```

Multi Cores + Multi Threads

Demo xử lý song song với MultiThreads + MultiCores



Tài liệu tham khảo

1. http://www.tutorialspoint.com/java/java_thread_synchronization.htm
2. http://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html