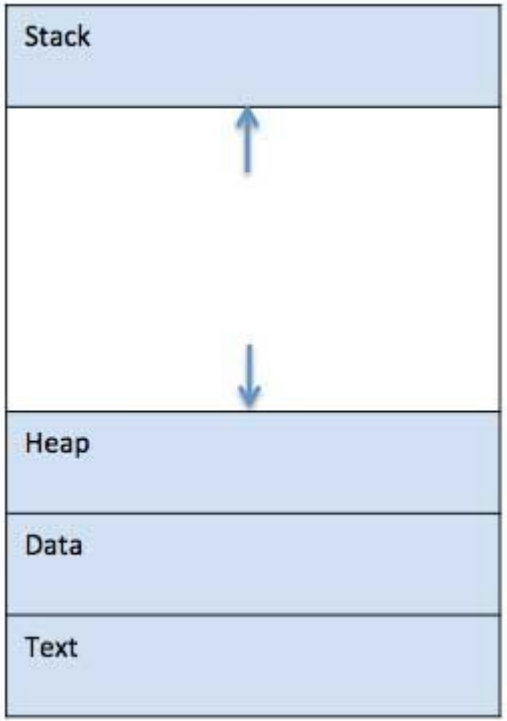
Multi Threading in Java

Concurrent Programming

- In concurrent programming, there are two basic units of execution: ***processes*** and ***threads***. In the Java programming language, concurrent programming **is mostly concerned with threads.**

Processes

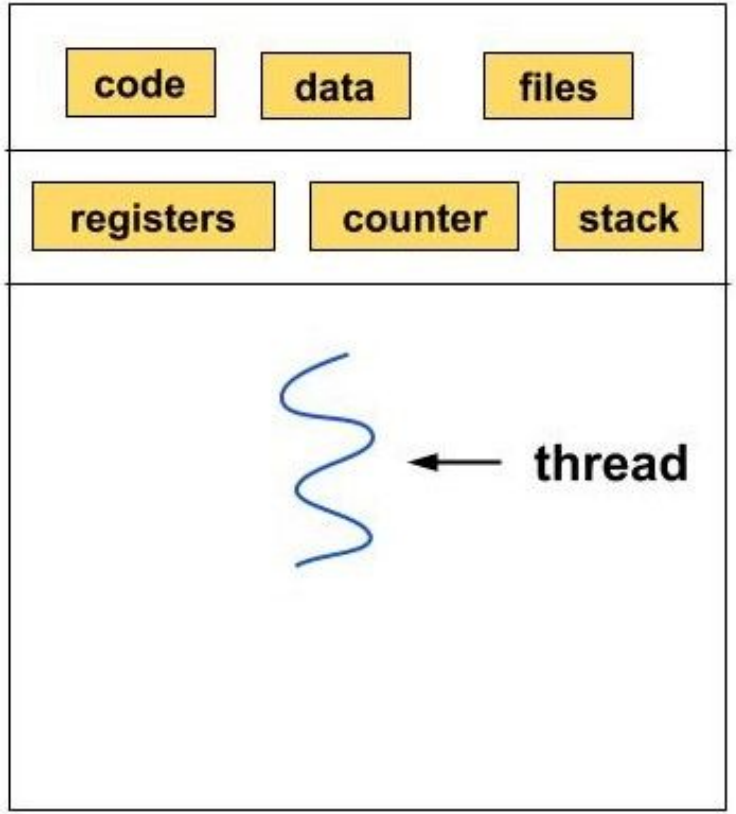
- A process has a self-contained execution environment. A process generally has **a complete, private set of basic run-time resources**; in particular, **each process has its own memory space.**



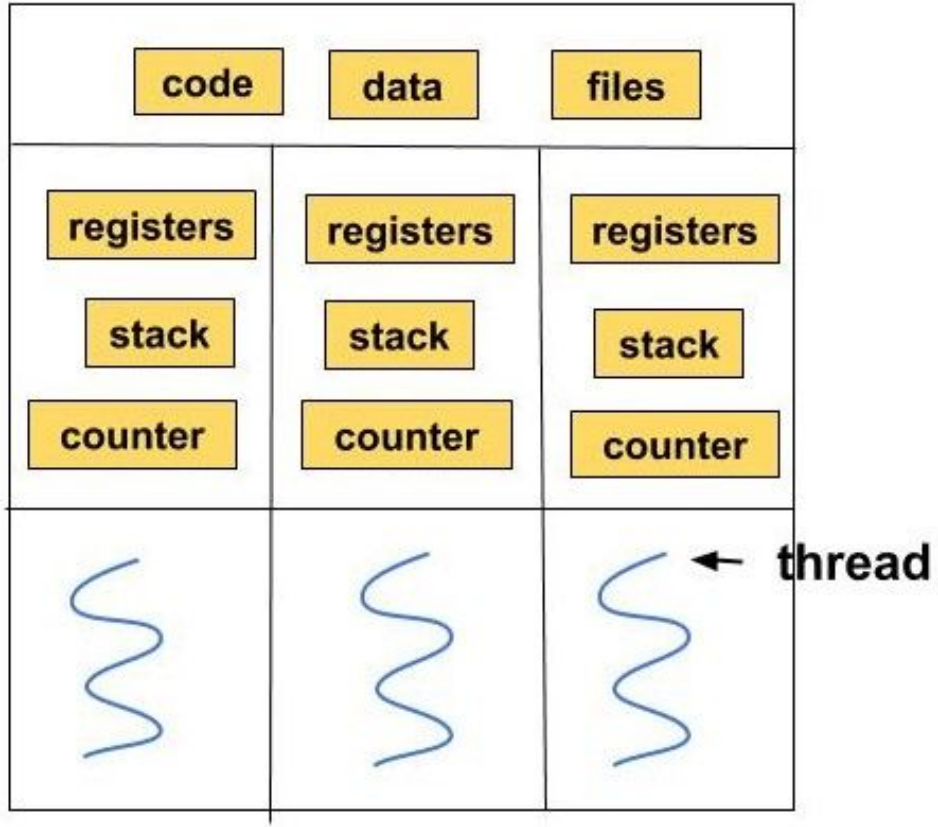
Source: https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

Threads

- Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files.



Single-threaded process



Multi-threaded process

Different between process & thread

- **Ex:** Open a browser (process) and a new tab (thread).
- **Definition:** Process means a program that is currently under execution, whereas thread is an entity that resides within a process that can be scheduled for execution.
- **Resources:** Processes are also called **heavyweight processes** as they use more resources. The threads are called **lightweight processes** as they share resources.
- **Creation Time:** The process creation time takes more time as compared to thread creation time.

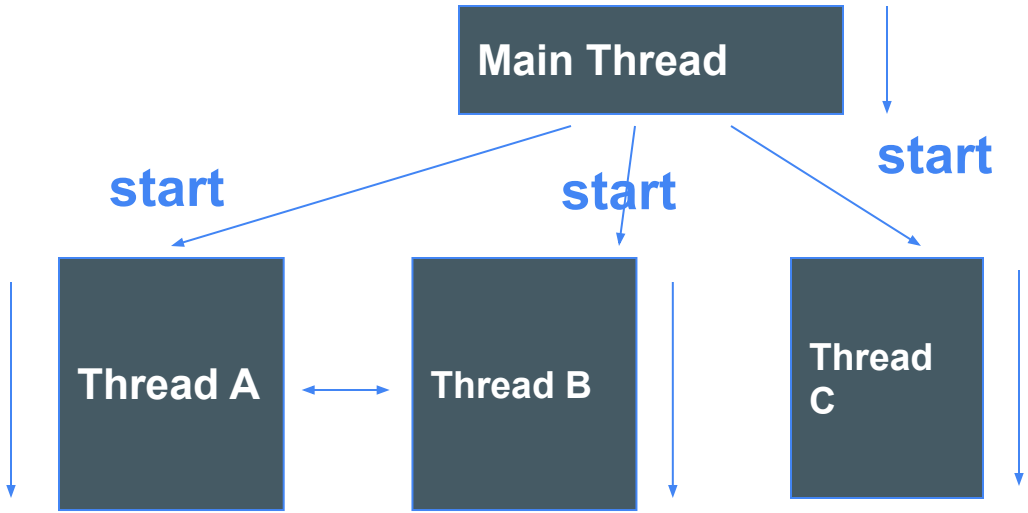
Source: https://afteracademy.com/blog/what-is-a-thread-in-os-and-what-are-the-differences-between-a-process-and-a-thread

Different between process & thread

- **Memory:** A Process is run in separate memory space, whereas threads run in shared memory space.
- **Sharing Data:** Different processes have different copies of data, files, and codes whereas threads share the same copy of data, file and code segments.
- **Communication:** The communication between threads requires less time as compared to the communication between processes.

Source: https://afteracademy.com/blog/what-is-a-thread-in-os-and-what-are-the-differences-between-a-process-and-a-thread

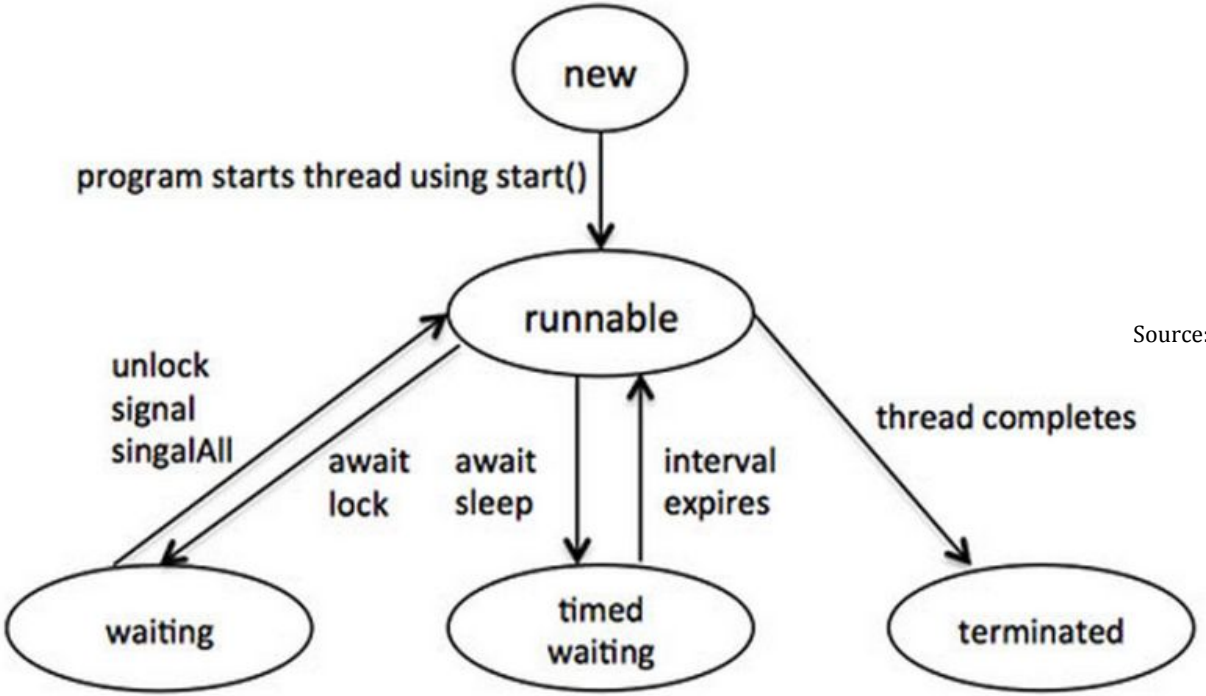
Multi threads

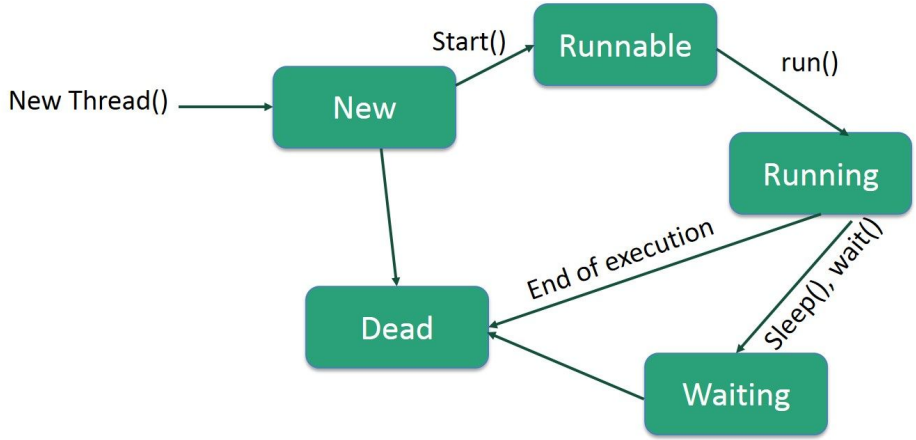


Các thread có thể chuyển đổi dữ liệu với nhau

- 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.

Life cycle of thread





Source: https://www.tutorialspoint.com/java/java_multithreading.htm

Life cycle of 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.

Priority

- 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.

Defining & Starting a Thread

- An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this

Defining & Starting a Thread

```
public class HelloRunnable
implements Runnable {
   public void run() {
      System.out.println("Hello from a
      thread!");
   }
   public static void main(String args[]){
      (new Thread(new
      HelloRunnable())).start();
   }
}
```

```
public class HelloThread extends
Thread {
   public void run() {
      System.out.println("Hello from a
      thread!");
   }
   public static void main(String args[]) {
      (new HelloThread()).start();
   }
}
```

Defining & Starting a Thread using 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 ();
      }
   }

}
```

Defining & Starting a Thread using 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.
```

Defining & Starting a Thread using Thread class

- **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 ();
       }
    }

}
```

Defining & Starting a Thread using Thread class

```
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);
```

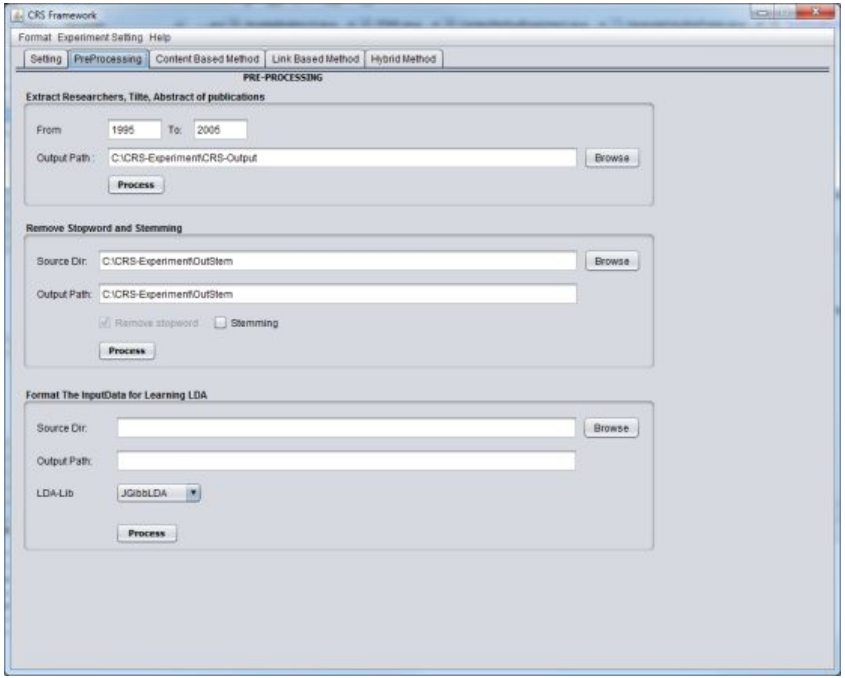
Multi Cores + Multi Threads

- 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



References

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

Homework

- Viết các chương trình bên dưới dùng multithread để xử lý song song và đo thời gian thực hiện tính toán so với single thread.

Bài 1: Viết chương trình đọc 1 file text, tách các câu trong file (mỗi câu cách nhau bởi các dấu "." ";", "!", "?") và ghi ra các câu đọc được ra một file text mới theo định dạng là mỗi dòng 1 câu.

Bài 2: Viết chương trình kiểm tra xem 1 từ có xuất hiện trong file text hay không và đếm số lần xuất hiện của từ đó.

Bài 3: Viết chương trình đếm số lần xuất hiện của các từ khác nhau trong một file text.

Input: một hay nhiều files dạng văn bản

Output: từ điển gồm các từ khác nhau và tần suất xuất hiện của chúng, dạng HashMap [key, value]

Bài 4. Viết chương trình loại bỏ các stop words ra khỏi một câu văn bản tiếng Việt.

- Input: một câu + 1 file chứa các stop words
- Output: Các từ khác nhau trong câu và không có tính stop word. Mỗi từ quy ước cách nhau bằng khoảng trắng

Homework

Bài 5: Viết chương trình mã hóa một văn bản thành một vector n chiều

- Input: kho dữ liệu là một thư mục chứa nhiều file dạng văn bản
- Output: Mỗi file được biểu diễn bằng 1 vector n chiều.

(n: là số từ khác nhau trong kho dữ liệu; giá trị mỗi chiều là 1 nếu từ đó tồn tại trong file, ngược lại là 0)

Bài 6: Viết chương trình mã hóa một văn bản thành một vector n chiều, với trọng số của mỗi chiều là tf*idf (tf: term frequency; idf; inverse document frequency)

- Input: kho dữ liệu là một thư mục chứa nhiều file dạng văn bản
- Output: Mỗi file được biểu diễn bằng 1 vector n chiều.

(n: là số từ khác nhau trong kho dữ liệu; giá trị mỗi chiều là tf*idf)

Bài 7. Viết chương trình chat qua LAN dùng Java Socket