Kỹ thuật xây dựng ứng dụng phía server

Nội dung

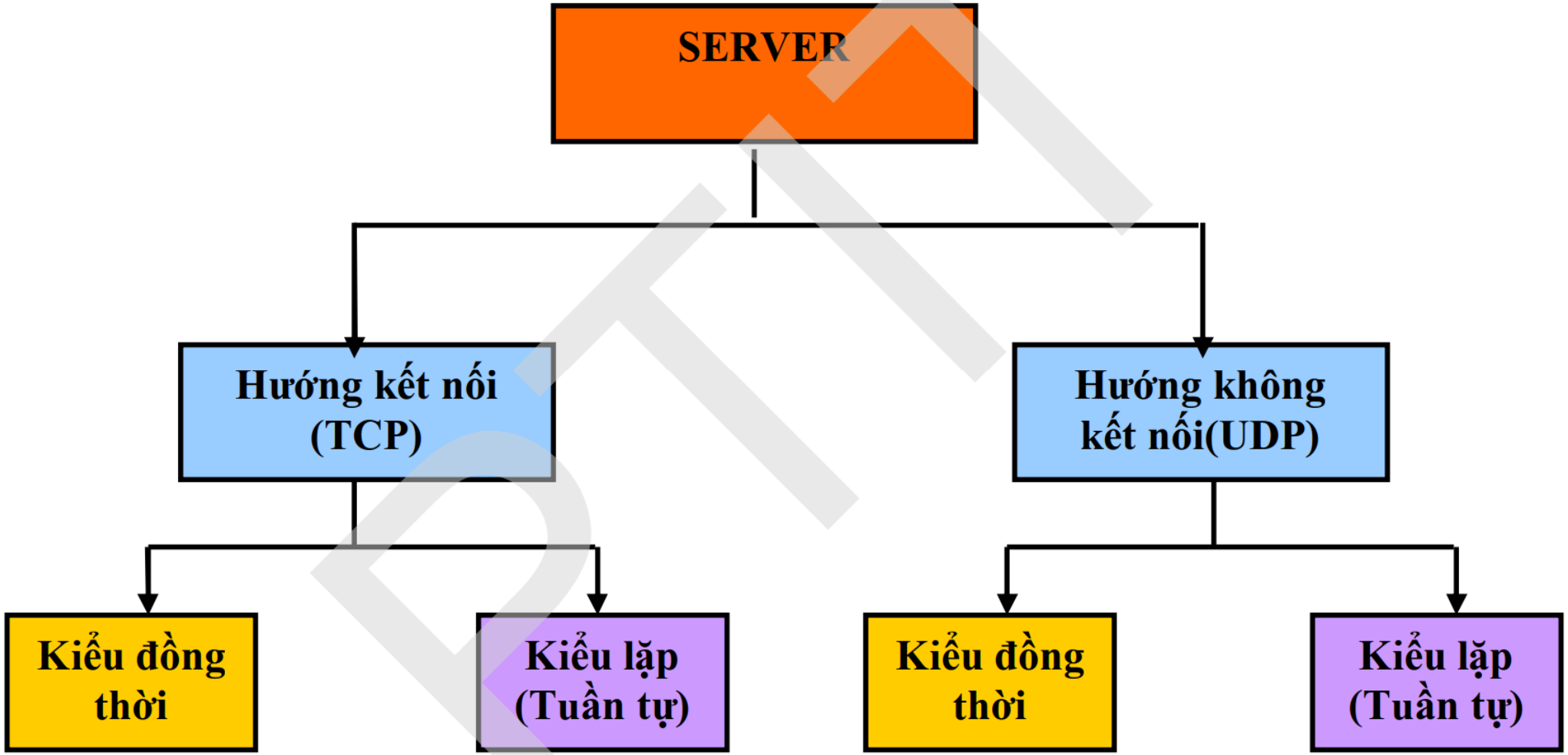
- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

Giới thiệu

- Với mô hình client-server, 1 server có thể phục vụ 1 hoặc nhiều client theo kiểu đồng thời (concurrent) hoặc lặp (iterative)
  - TCPSocket:
    - ✓ Iterative server
    - ✓ Concurrent server: phổ biến
  - UDPSocket
    - ✓ Iterative server: phổ biến
    - ✓ Concurrent server

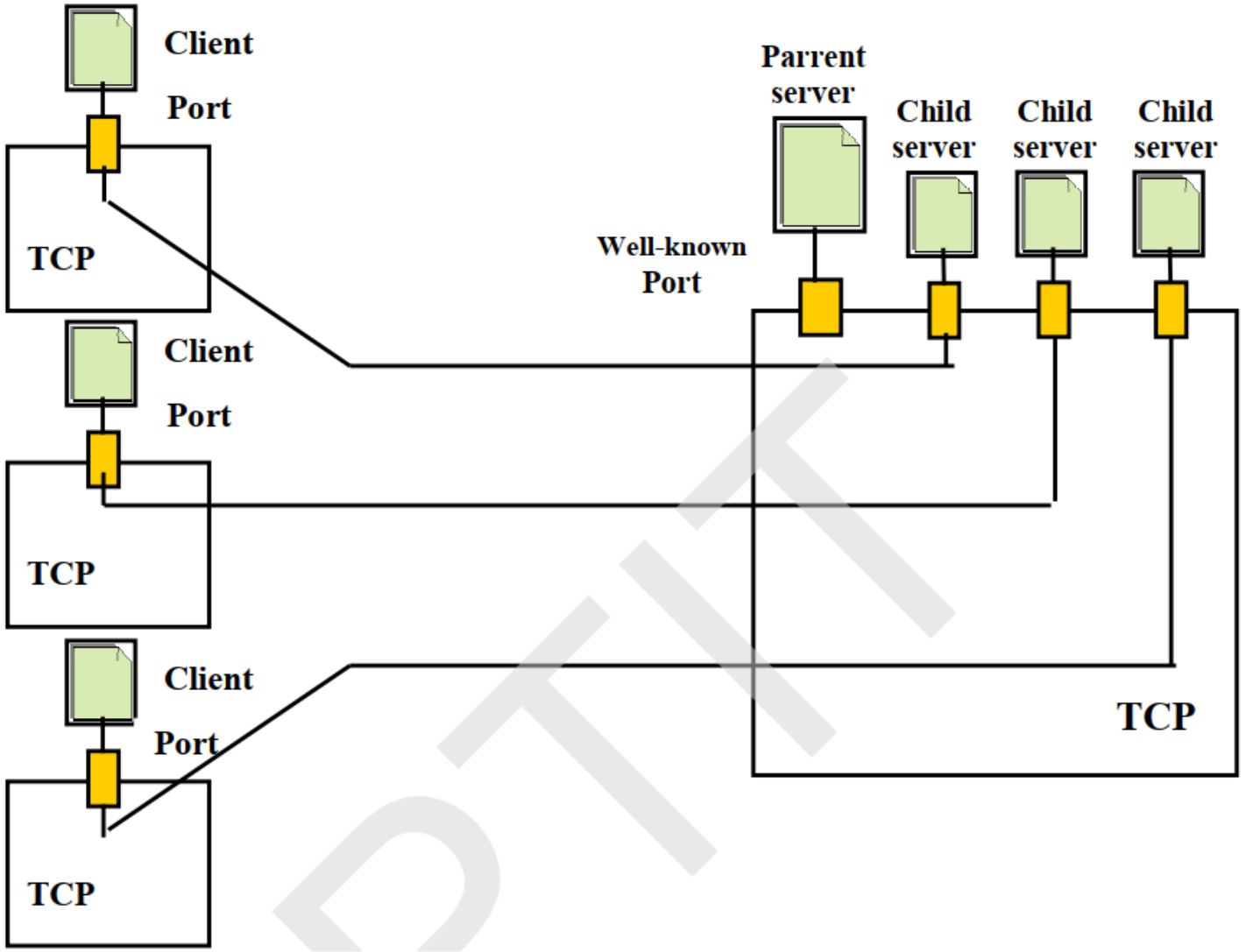

SERVER
Hướng kết nối (TCP)
Hướng không kết nối(UDP)
Kiểu đồng thời
Kiểu lặp (Tuần tự)
Kiểu đồng thời
Kiểu lặp (Tuần tự)

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

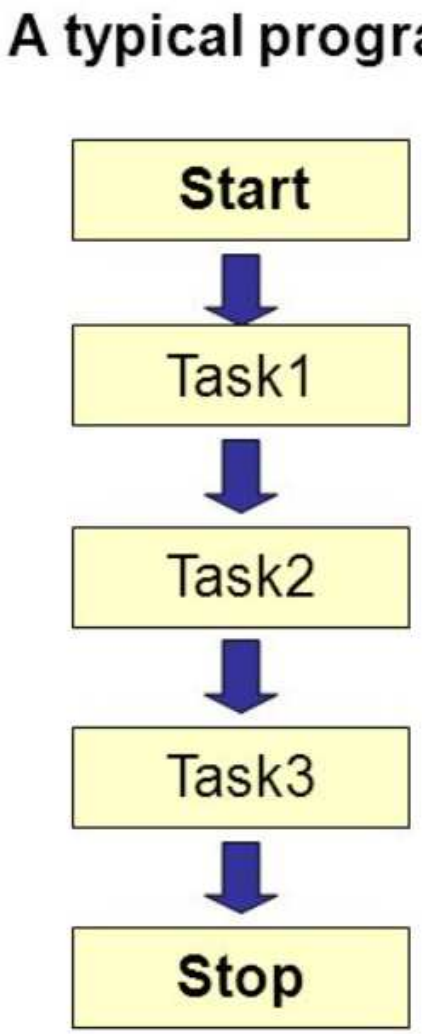
Concurrent TCPServer

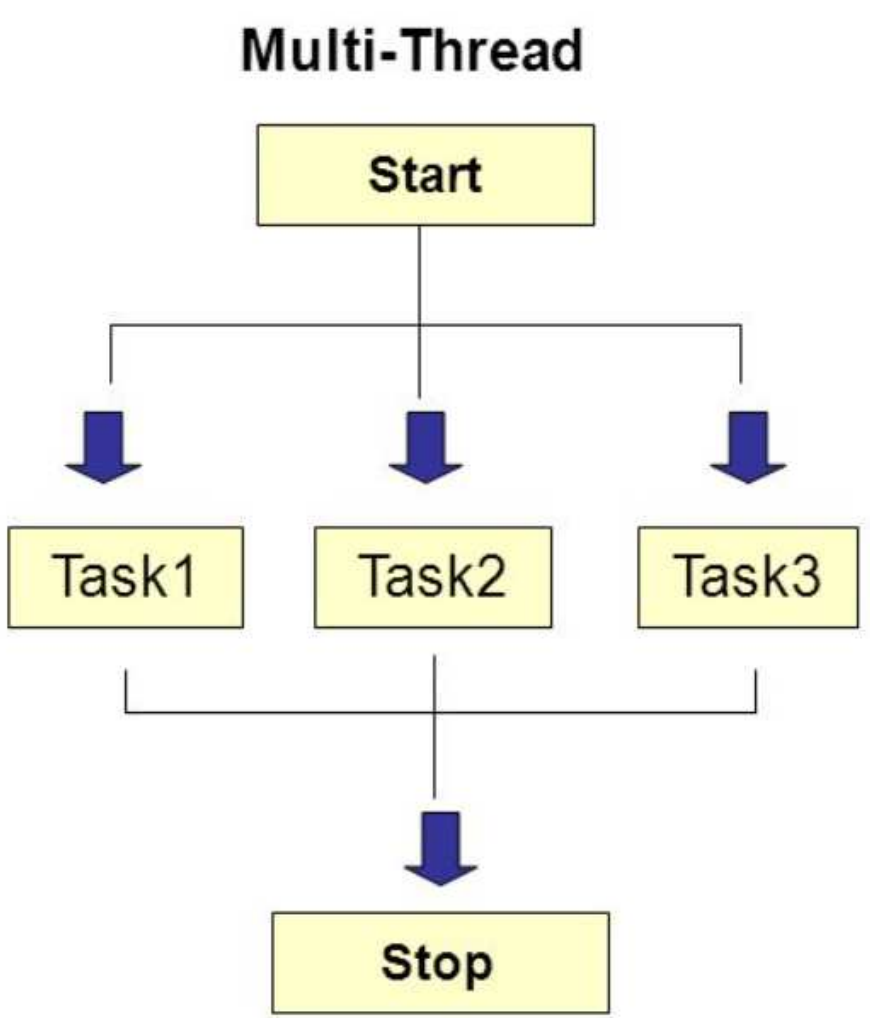
- Mở 1 cổng quy ước khi khởi tạo để lắng nghe tín hiệu từ client.
- Tạo server con và mở cổng phụ để kết nối đến mỗi client.
- Server chính tiếp tục lắng nghe tại cổng quy ước



Concurrent TCPServer

- Hai kỹ thuật:
  - Chương trình đa tiến trình (process): sinh ra tiến trình con khi có client kết nối → context switch?
  - Chương trình đa luồng (tiểu trình, thread): sử dụng hiệu quả tài nguyên, đặc biệt là CPU.

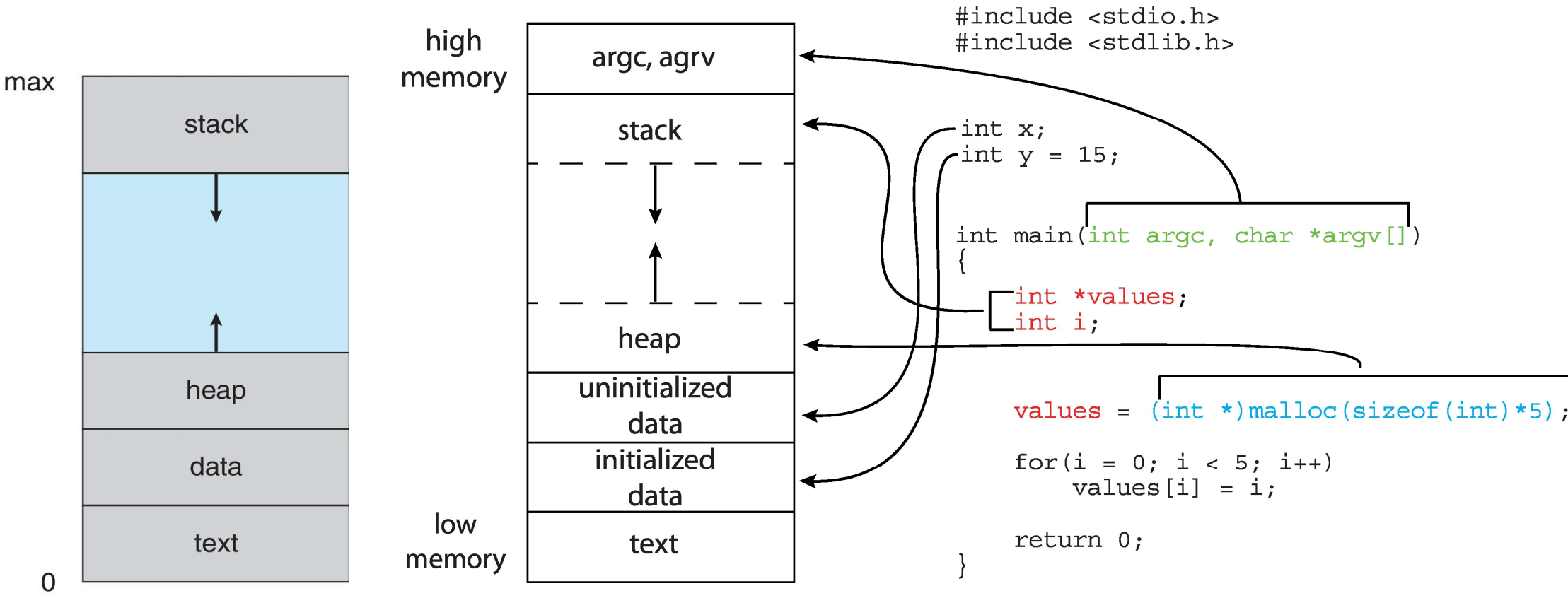




Nhắc lại kiến thức Hệ điều hành

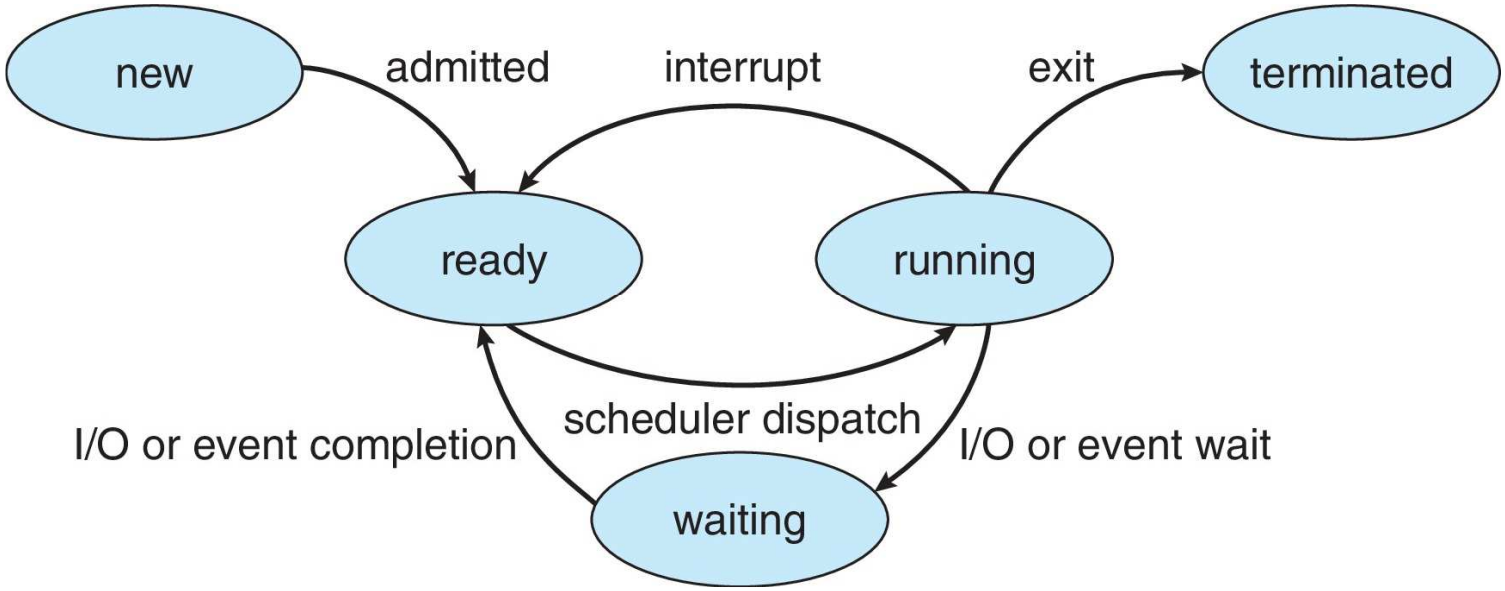
- Program *vs* Process
- Process: a program in execution; process execution must progress in sequential fashion. No parallel execution of instructions of a single process
  - The program code, also called text section
  - Current activity including program counter, processor registers
  - Stack containing temporary data: function parameters, return addresses, local variables
  - Data section containing global variables
  - Heap containing memory dynamically allocated during run time
- Program is passive entity stored on disk (executable file); process is active. Program becomes process when an executable file is loaded into memory

Nhắc lại kiến thức Hệ điều hành



Memory layout of a C program

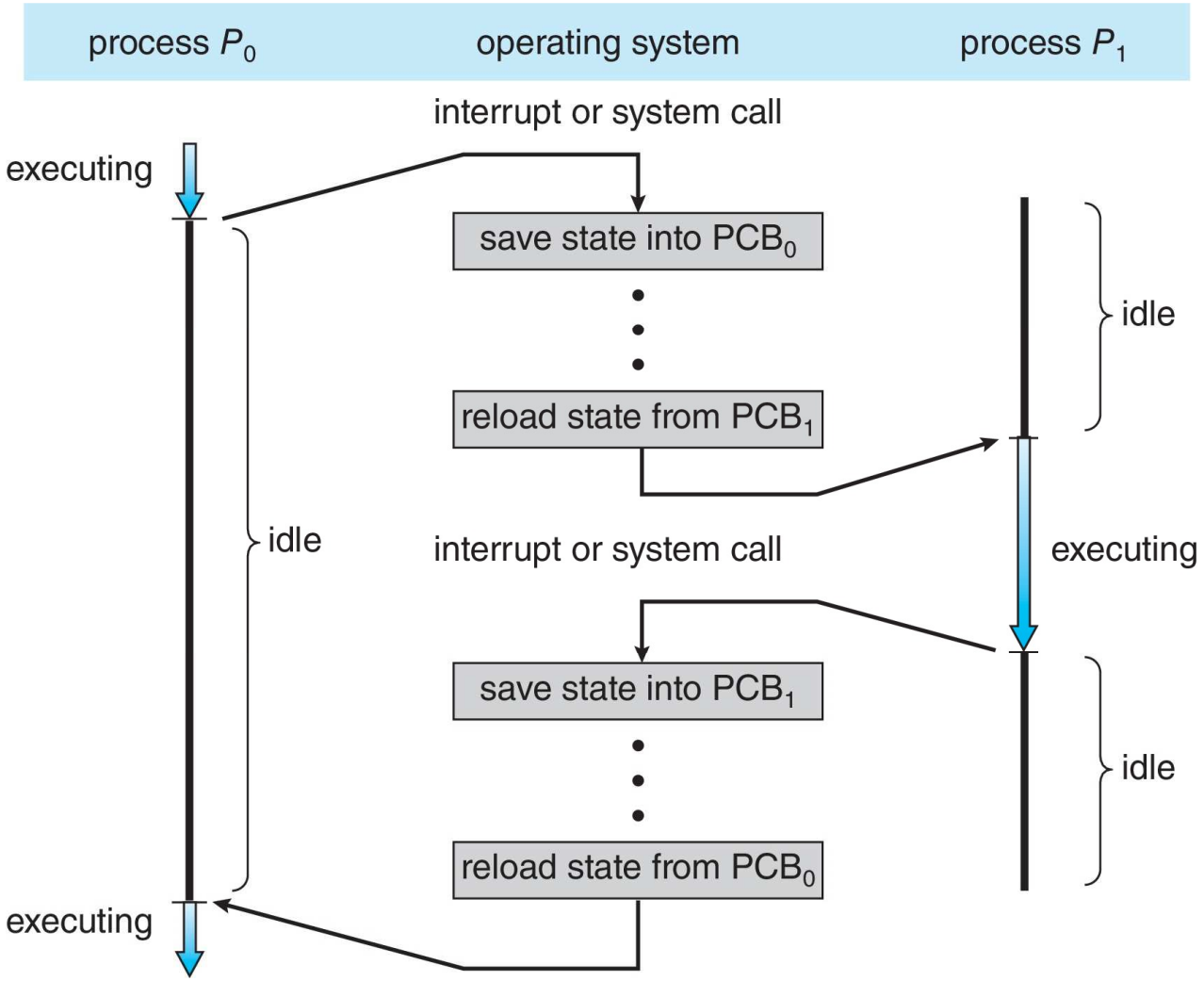
- Process State & Process Control Block (PCB)



PCB

Nhắc lại kiến thức Hệ điều hành

- Context Switch



Nhắc lại kiến thức Hệ điều hành

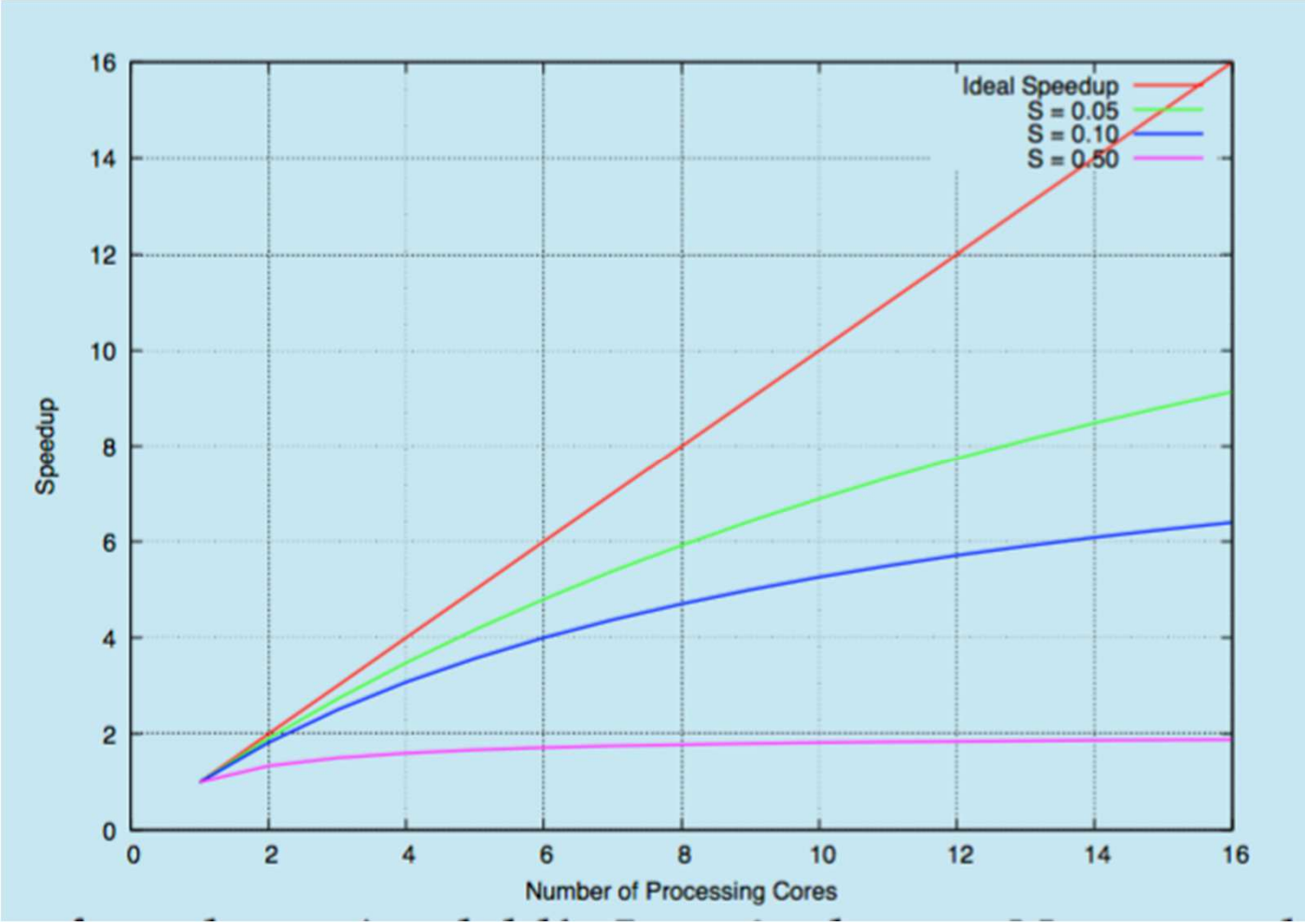
- Interprocess Communication (IPC):
  - Shared Memory
  - Message passing (Message queue)
  - Pipe
- Process vs Thread
- Thread motivation:
  - Most modern applications are multithreaded
  - Threads run within application
  - Multiple tasks with the application can be implemented by separate threads
  - Process creation is heavy-weight while thread creation is light-weight
  - Can simplify code, increase efficiency

- Amdahl's Law

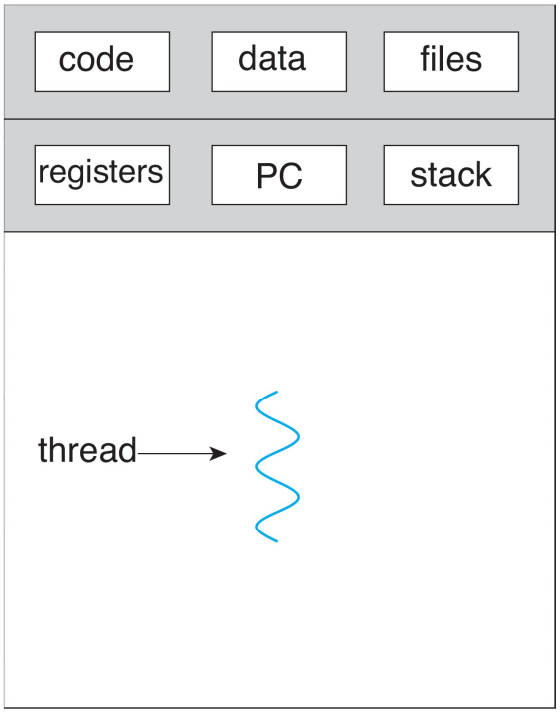
$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

  - *S* is serial portion
  - *N* processing cores

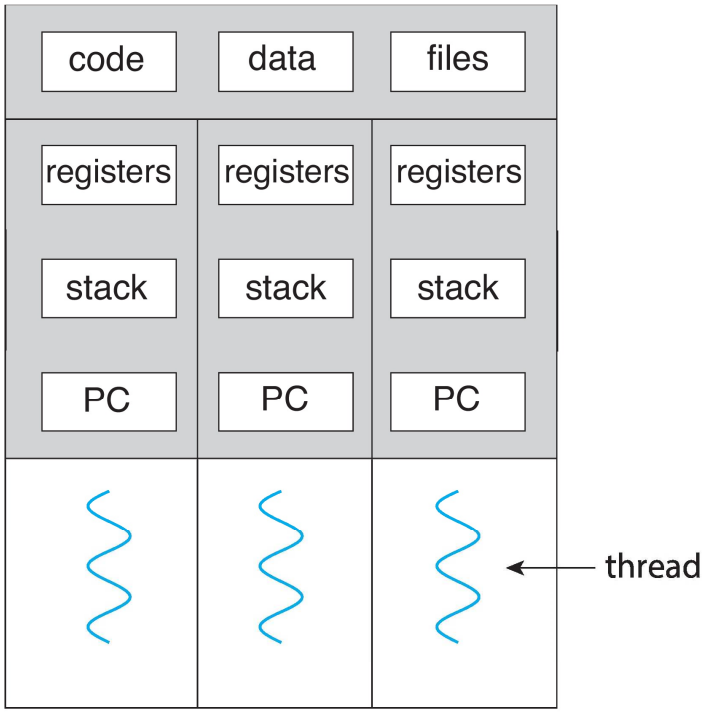
- N → ∞, *speedup* → $\frac{1}{S}$



- Single and Multithreaded processes

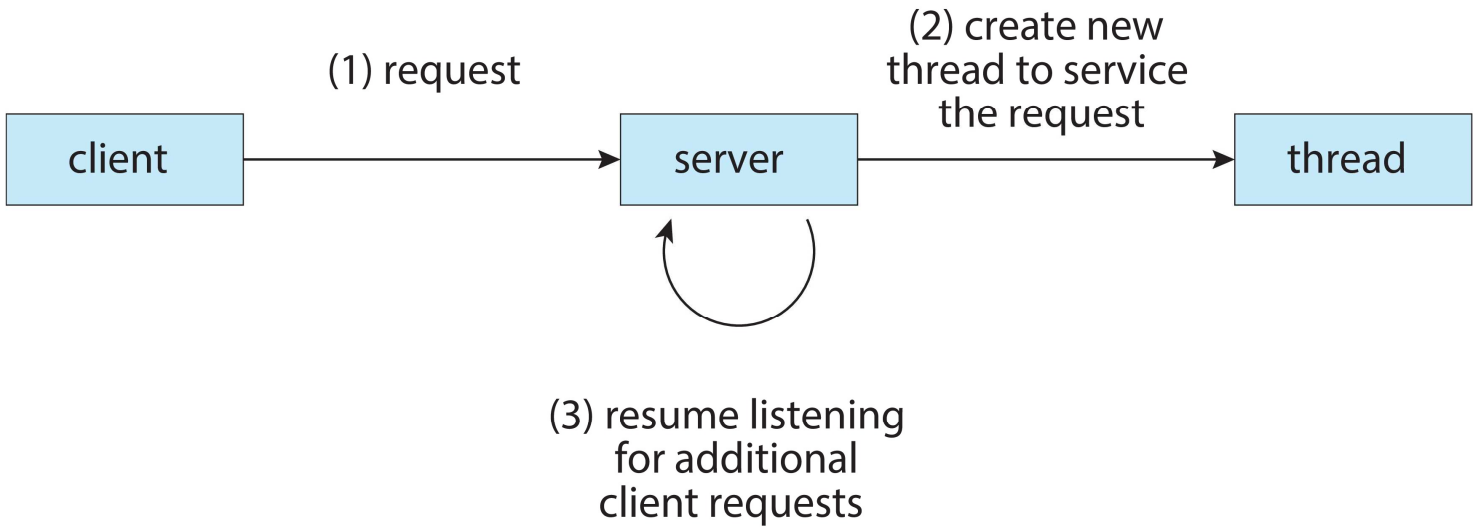


single-threaded process



multithreaded process

- Multithreaded server architecture



Concurrent TCPServer

- Thread (luồng): một chuỗi các lệnh được lập trình nhỏ nhất để có thể được quản lý độc lập bởi một bộ định thời, thường là một phần của hệ điều hành.
- Ưu điểm của đa luồng (multithread)?
- Các vấn đề lưu ý khi sử dụng multithread: synchronized, deadlock
- Multithread program gồm:
  - Main thread: khởi chạy đầu tiên, sinh ra các thread con, kết thúc sau cùng
  - Child thread
- Tạo thread trong Java:
  - Extends Thread class
  - Implement Runnable interface

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

Extends Thread

- Extends Thread class:

```
 3 class Student extends Thread {
 4     String fullName;
 5     int age;
 6     public Student(String fN, int age) {
 7         this.fullName = fN; this.age = age;
 8     }
 9     public void run() {
10         for(int i = 0 ; i<=age; i++)
11             System.out.println("My name is " + fullName + " - i=" + i + "/" + age);
12     }
13 }
14
15 public class Classroom {
16     public static void main(String[] args) {
17         Student s1 = new Student("Anh",25);
18         Student s2 = new Student("Bao",18);
19         Student s3 = new Student("Can",30);
20         s1.start();
21         s2.start();
22         s3.start();
23     }
24 }
```

```
My name is Anh - i=0/25
My name is Can - i=0/30
My name is Can - i=1/30
My name is Bao - i=0/18
My name is Can - i=2/30
My name is Can - i=3/30
My name is Can - i=4/30
My name is Can - i=5/30
My name is Anh - i=1/25
My name is Can - i=6/30
My name is Bao - i=1/18
```

Implements Runnable

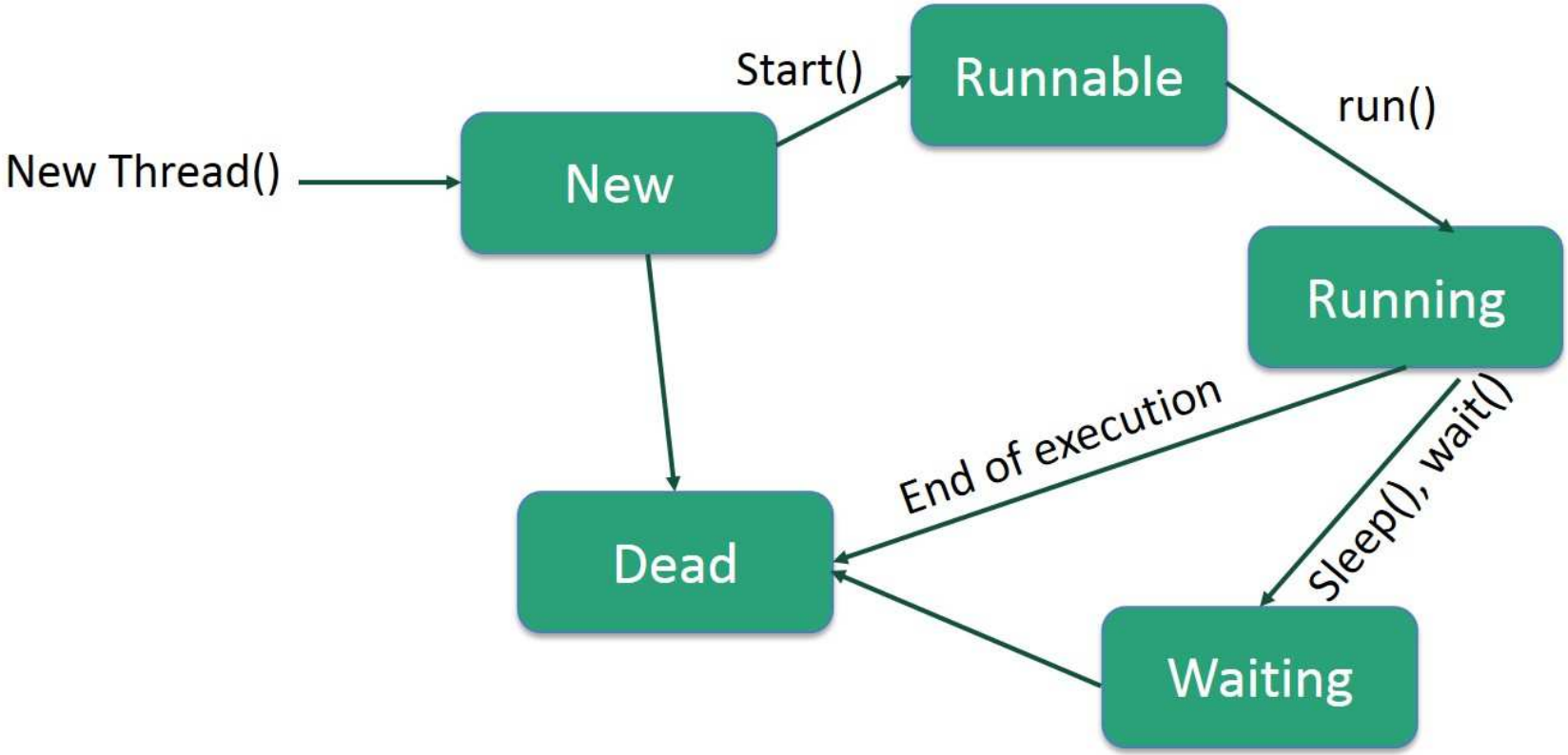
- Implements Runnable:

```
 3 class Student implements Runnable {
 4     String fullName;
 5     int age;
 6     public Student(String fN, int age) {
 7         this.fullName = fN; this.age = age;
 8     }
 9     public void run() {
10         for(int i = 0 ; i<=age; i++)
11             System.out.println("My name is " + fullName + " - i=" + i + "/" + age);
12     }
13 }
14
15 public class Classroom {
16     public static void main(String[] args) {
17         Student s1 = new Student("Anh",25);
18         Student s2 = new Student("Bao",18);
19         Thread t1 = new Thread(s1);
20         Thread t2 = new Thread(s2);
21         t1.start();
22         t2.start();
23     }
```

```
My name is Bao - i=0/18
My name is Bao - i=1/18
My name is Anh - i=0/25
My name is Bao - i=2/18
My name is Anh - i=1/25
My name is Bao - i=3/18
My name is Anh - i=2/25
My name is Bao - i=4/18
My name is Anh - i=3/25
```

Implements Runnable

- Ưu điểm Implement Runable interface:
  - Không cần kế thừa từ lớp Thread
  - ThreadPool

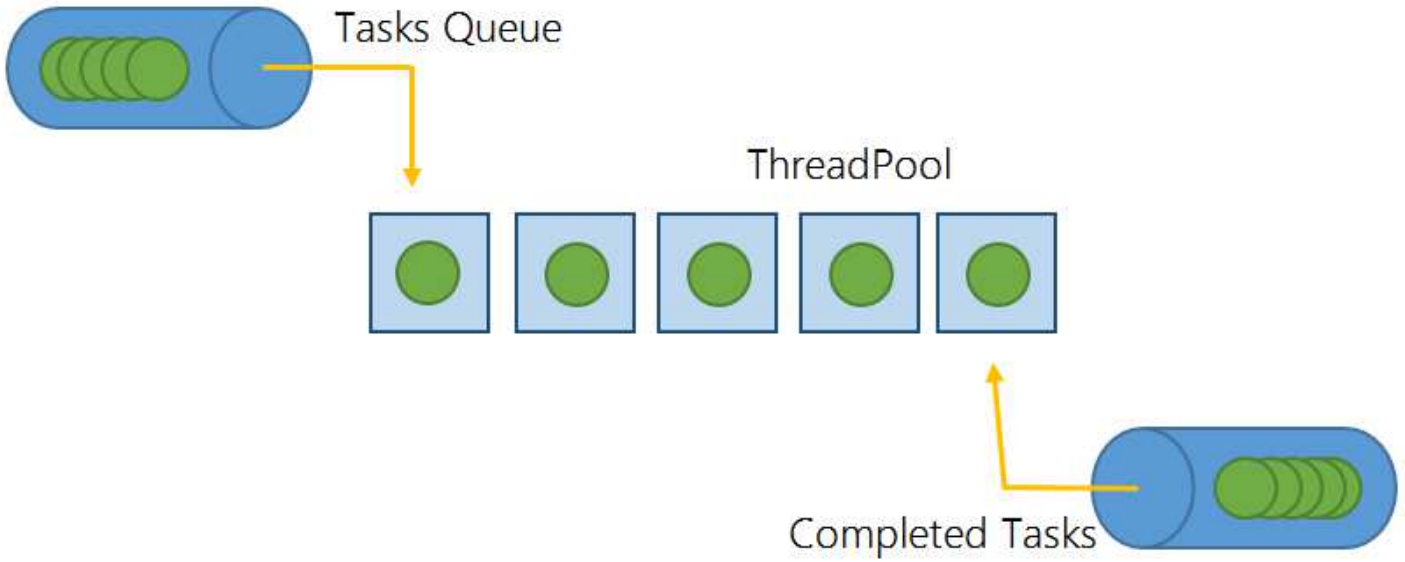


Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

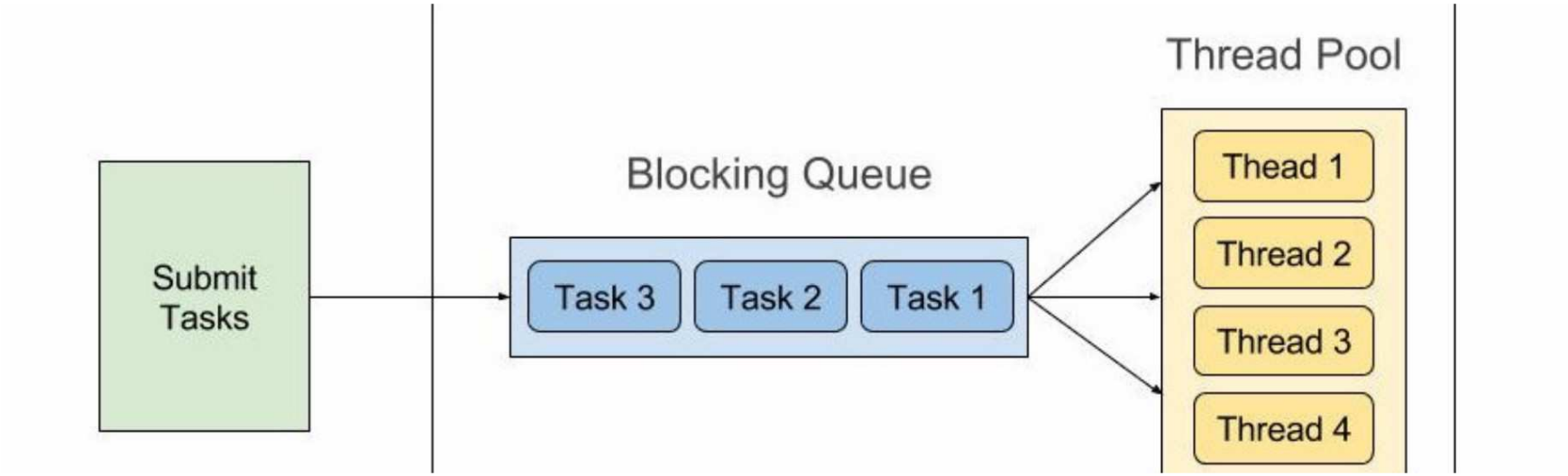
Thread Pool

- Tạo nhiều thread giải quyết công việc: số lượng thread VS hiệu năng VS cấp phát dữ liệu → Thread pool
- Thread pool: nhóm các thread đang chờ đợi công việc và tái sử dụng được nhiều lần.
- Ưu điểm: hiệu năng tốt hơn, tiết kiệm thời gian vì không cần phải tạo thread mới.



Thread Pool

- Nếu request > Thread pool → Blocking queue



Thread Pool

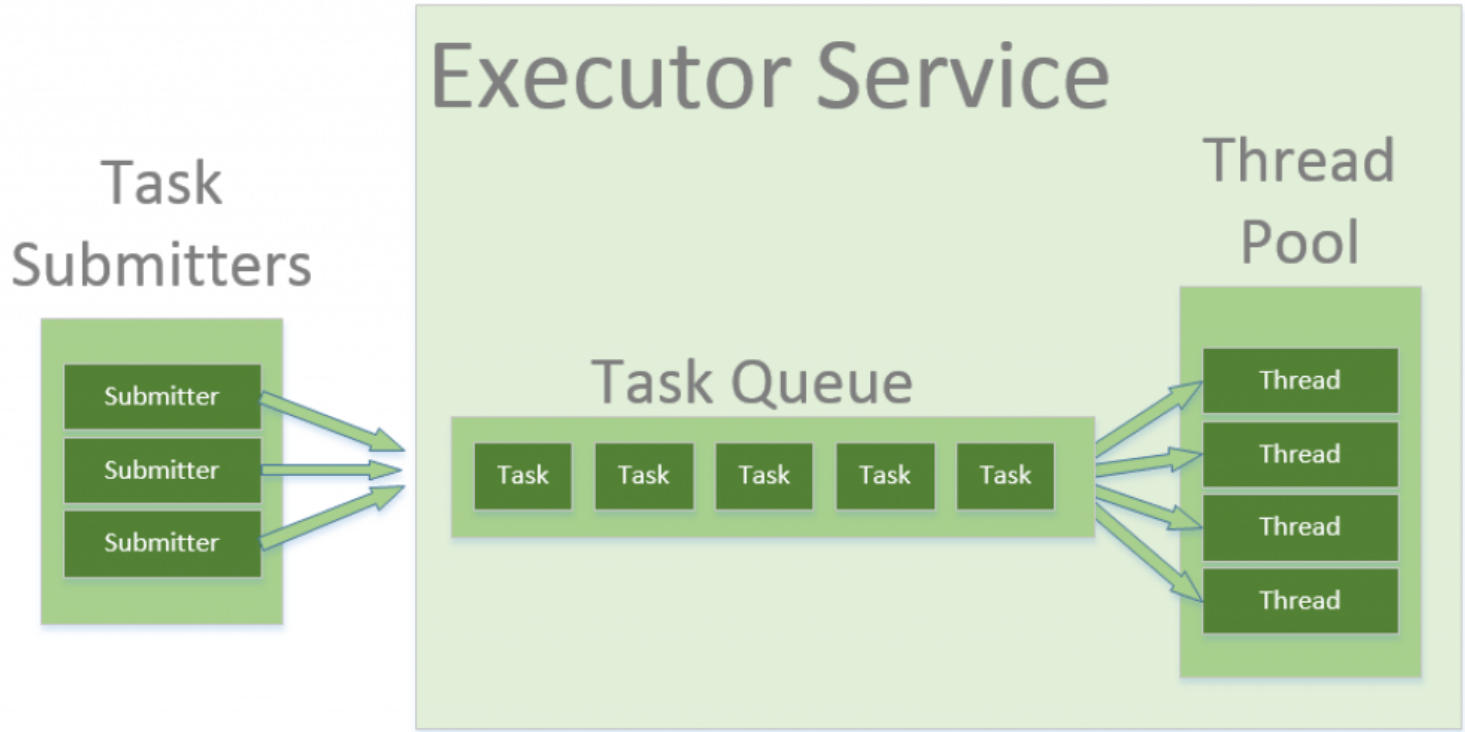
- Java Concurrency API hỗ trợ một vài loại ThreadPool sau:
  - Cached thread pool: mỗi task sẽ tạo ra thread mới nếu cần, nhưng sẽ tái sử dụng lại các thread cũ
  - Fixed thread pool: giới hạn số lượng tối đa của các Thread được tạo ra. Các task khác đến sau phải chờ trong hàng đợi (BlockingQueue)
  - Single-threaded pool: chỉ giữ một Thread thực thi một nhiệm vụ một lúc.
  - Fork/Join pool: một Thread đặc biệt sử dụng Fork/Join Framework bằng cách tự động chia nhỏ công việc tính toán cho các core xử lý

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

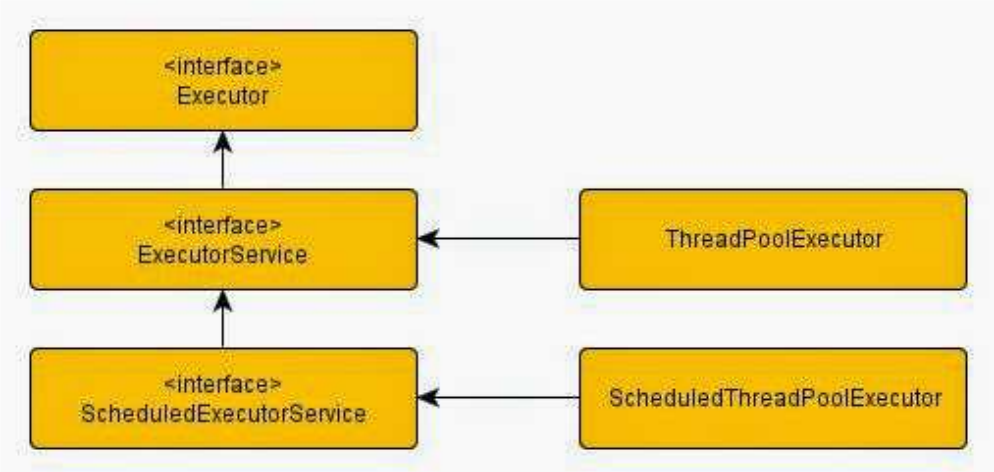
Executor là gì

- Executor là một đối tượng chịu trách nhiệm quản lý các luồng và thực hiện các tác vụ Runnable được yêu cầu xử lý.
- Executor tách riêng các chi tiết của việc tạo Thread, lập lịch (scheduling), ... nhằm tập trung phát triển logic của tác vụ mà không quan tâm đến các chi tiết quản lý Thread.



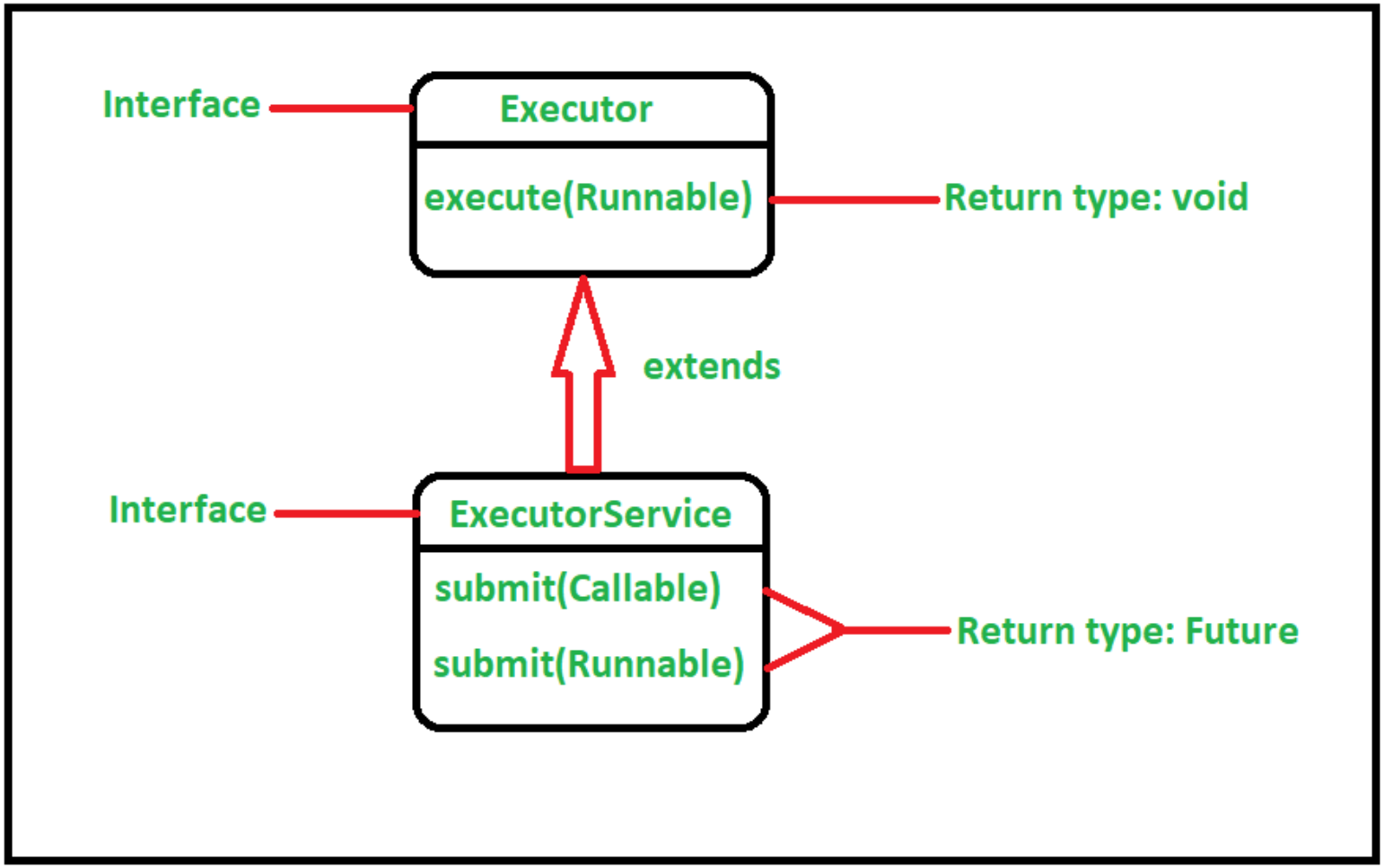
Executor framework (java.util.concurrent.ExecutorService)

- "Executor framework" trong gói java.util.concurrent giúp tạo và quản lý các "ThreadPool" và "Thread Factories".
  - Executor: là interface cha của tất cả các Executor, method: execute(Runnable)
  - ExecutorService: theo dõi tiến trình của các tác vụ trả về giá trị (Callable) thông qua đối tượng Future, và quản lý việc kết thúc các luồng. Method: submit() và shutdown().
  - ScheduledExecutorService: là một ExecutorService có thể lên lịch cho các tác vụ để thực thi sau một khoảng thời gian nhất định, hoặc để thực hiện định kỳ. Method: schedule(), scheduleAtFixedRate() & scheduleWithFixedDelay()



Executor framework (java.util.concurrent.ExecutorService)

- ExecutorService



Executor framework

- Tạo Executor:
  - newSingleThreadExecutor(): ThreadPool chỉ có 1 thread và task thực thi tuần tự.
  - newCachedThreadPool(): ThreadPool có nhiều Thread và các task sẽ được xử lý song song. Các Thread cũ sau khi xử lý xong sẽ được sử dụng lại cho task mới. Mặc định, Thread không được sử dụng trong vòng 60 giây thì sẽ bị tắt.
  - newFixedThreadPool(int nThreads): ThreadPool chứa tối đa nThreads. Khi Pool đạt đến giá trị tối đa nThreads, các Thread còn lại sẽ được đưa vào Blocking Queue.
  - newScheduledThreadPool(int corePoolSize): tương tự như newCachedThreadPool() nhưng sẽ có thời gian delay giữa các Thread.
  - newSingleThreadScheduledExecutor(): tương tự như newSingleThreadExecutor() nhưng sẽ có khoảng thời gian delay giữa các Thread.

Ví dụ ExecutorService

- Worker Thread

```
 5 public class Worker implements Runnable {
 6     private String task;
 7     public Worker(String s) {
 8         this.task = s;
 9     }
10     public void run() {
11         System.out.println(Thread.currentThread().getName() + " starting. Task = " + task);
12         try {
13             Random rand = new Random();
14             Thread.sleep(rand.nextInt(3000)+500); // sleep between 500 - 3500 ms
15         } catch (InterruptedException e) {e.printStackTrace();}
16         System.out.println(Thread.currentThread().getName() + " finished.");
17     }
18 }
```

Ví dụ ExecutorService – newSingleThreadExecutor()

- newSingleThreadExecutor()

```
 3 import java.util.concurrent.ExecutorService;
 4 import java.util.concurrent.Executors;
 5
 6 public class SingleThreadExecutor {
 7     public static void main(String[] args) {
 8         ExecutorService executor =
 9                 Executors.newSingleThreadExecutor();
10         for(int i=1; i<=7; i++) {
11             Runnable worker =
12                     new Worker(Integer.toString(i));
13             executor.execute(worker);
14         }
15         executor.shutdown();
16         // Wait until all threads are finish
17         while(!executor.isTerminated()) {}
18         System.out.println("All threads finished");
19     }
20 }
21
```

```
pool-1-thread-1 Starting. Task = 1
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 2
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 3
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 4
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 5
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 6
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 7
pool-1-thread-1 Finished.
All threads finished
```

Ví dụ ExecutorService – newFixedThreadPool(int nThreads)

- newFixedThreadPool(int nThreads)

```java
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

public class FixedThreadPool {
    public static void main(String[] args) {
        ExecutorService executor =
                Executors.newFixedThreadPool(3);
        for(int i=1; i<=7; i++) {
            Worker worker =
                    new Worker(Integer.toString(i));
            executor.execute(worker);
        }
        executor.shutdown();
        while(!executor.isTerminated()) {}
        System.out.println("All threads finished");
    }
}
```

```
pool-1-thread-1 Starting. Task = 1
pool-1-thread-3 Starting. Task = 3
pool-1-thread-2 Starting. Task = 2
pool-1-thread-3 Finished.
pool-1-thread-3 Starting. Task = 4
pool-1-thread-3 Finished.
pool-1-thread-3 Starting. Task = 5
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 6
pool-1-thread-1 Finished.
pool-1-thread-1 Starting. Task = 7
pool-1-thread-2 Finished.
pool-1-thread-3 Finished.
pool-1-thread-1 Finished.
All threads finished
```

Ví dụ ExecutorService – newCachedThreadPool()

- newCachedThreadPool()

```
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5
6  public class CachedThreadPool {
7      public static void main(String[] args) {
8          ExecutorService executor =
9                  Executors.newCachedThreadPool();
10         for(int i=1; i<=7; i++) {
11             Worker worker =
12                     new Worker(Integer.toString(i));
13             executor.execute(worker);
14         }
15         executor.shutdown();
16         while(!executor.isTerminated()) {}
17         System.out.println("All threads finished");
18     }
19 }
```

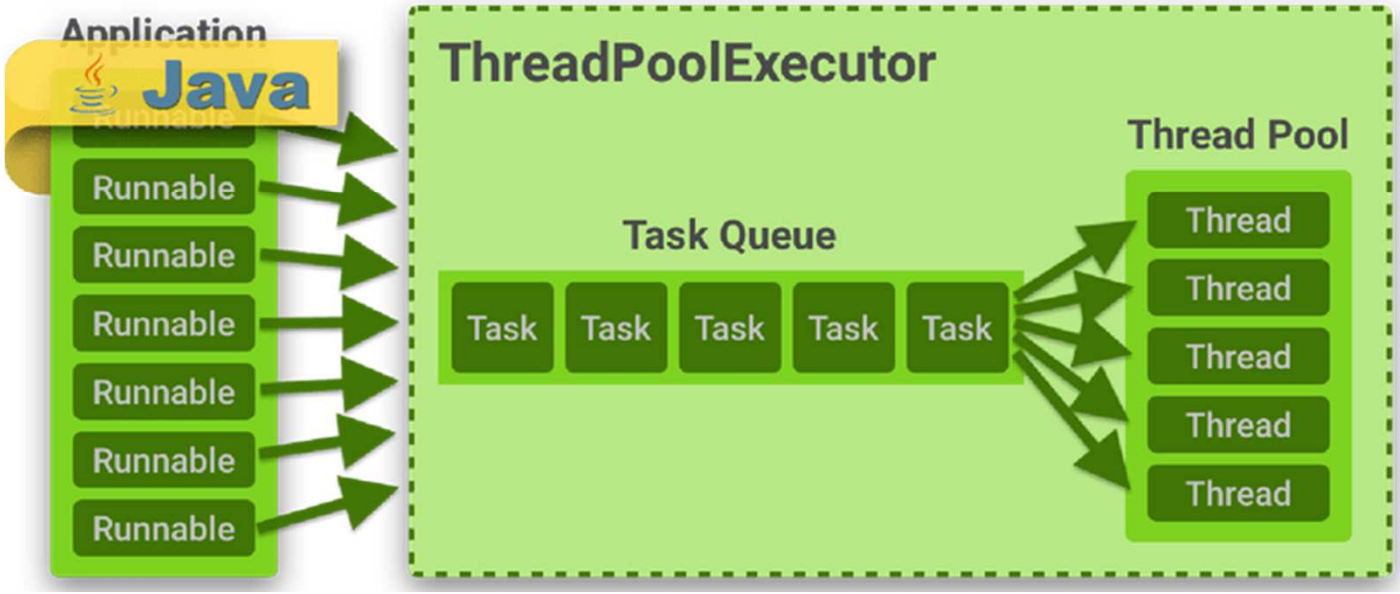
```
pool-1-thread-3 Starting. Task = 3
pool-1-thread-6 Starting. Task = 6
pool-1-thread-5 Starting. Task = 5
pool-1-thread-1 Starting. Task = 1
pool-1-thread-2 Starting. Task = 2
pool-1-thread-4 Starting. Task = 4
pool-1-thread-7 Starting. Task = 7
pool-1-thread-2 Finished.
pool-1-thread-4 Finished.
pool-1-thread-1 Finished.
pool-1-thread-5 Finished.
pool-1-thread-7 Finished.
pool-1-thread-3 Finished.
pool-1-thread-6 Finished.
All threads finished
```

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

ThreadPoolExecutor

- ThreadPool thông thường (ExecutorService) không đủ linh động theo tình huống: số lượng thread cố định hoặc cho phép tạo quá nhiều thread.
- ThreadPoolExecutor cho phép tùy biến số lượng Thread theo kịch bản.
  - corePoolSize: số lượng Thread mặc định trong Pool
  - maxPoolSize: số lượng tối đa Thread trong Pool
  - queueCapacity: số lượng tối đa của BlockingQueue



ThreadPoolExecutor

- Ví dụ ThreadPoolExecutor:
  - corePoolSize: 5
  - maxPoolSize: 15
  - queueCapacity: 100
- Giải thích:
  - Khi có request, ThreadPoolExecutor sẽ tạo trong Pool tối đa 5 thread (*corePoolSize*).
  - Khi số lượng thread vượt quá 5. ThreadPoolExecutor sẽ cho thread mới vào hàng đợi.
  - Khi số lượng hàng đợi full 100 (*queueCapacity*) → bắt đầu tạo thêm Thread mới.
  - Số Thread mới được tạo tối đa là 15 (*maxPoolSize*).
  - Khi Request vượt quá số lượng 15 thread. Request sẽ bị từ chối!

ThreadPoolExecutor

- Ví dụ ThreadPoolExecutor:

```
3  import java.util.concurrent.ArrayBlockingQueue;
4  import java.util.concurrent.ThreadPoolExecutor;
5  import java.util.concurrent.TimeUnit;
6
7  public class ThreadPoolExecutorExample {
8      public static void main(String[] args) {
9          int corePoolSize = 1;
10         int maximumPoolSize = 3;
11         int queueCapacity = 8;
12         ThreadPoolExecutor executor =
13                 new ThreadPoolExecutor(corePoolSize, maximumPoolSize, 10,
14                         TimeUnit.SECONDS, new ArrayBlockingQueue<>(queueCapacity));
15         for (int i = 1; i <= 10; i++) {
16             executor.execute(new Worker("Request-" + i));
17         }
18         executor.shutdown();
19         while (!executor.isTerminated()) {}
20         System.out.println("All threads finished");
21     }
22 }
```

Lưu ý khi sử dụng ExecutorService

- Phương thức shutdown(): ExecutorService sẽ từ chối nhận thêm các task, tất cả các task được thêm vào trước khi gọi shutdown() đều sẽ được thực thi, các task thêm sau sẽ bị từ chối (rejected).
- Phương thức shutdownNow(): tắt ExecutorService ngay lập tức, các task trong Queue cũng sẽ bị loại bỏ.

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

Ví dụ TCPSocket multithread client -> server, Worker Thread

```
10 public class Worker implements Runnable {
11     private Socket socket;
12     public Worker(Socket s) {
13         this.socket = s;
14     }
15     public void run() {
16         System.out.println("Client " + socket.toString() + " accepted");
17         try {
18             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
19             BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
20             String input = "";
21             while(true) {
22                 input = in.readLine();
23                 System.out.println("Server received: " + input + " from " + socket.toString());
24                 if(input.equals("bye"))
25                     break;
26             }
27             System.out.println("Closed socket for client " + socket.toString());
28             in.close();
29             out.close();
30             socket.close();
31         } catch (IOException e) {
32             System.out.println(e);
33         }
34     }
35 }
```

Ví dụ TCPSocket multithread client -> server, Server code

```java
public class Server {
    public static int port = 1234;
    public static int numThread = 2;
    private static ServerSocket server = null;

    public static void main(String[] args) throws IOException {
        ExecutorService executor = Executors.newFixedThreadPool(numThread);
        try {
            server = new ServerSocket(port);
            System.out.println("Server binding at port " + port);
            System.out.println("Waiting for client...");
            while(true) {
                Socket socket = server.accept();
                executor.execute(new Worker(socket));
            }
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            if(server!=null)
                server.close();
        }
    }
}
```

Ví dụ TCPSocket multithread client -> server, Client code

```
15    public static void main(String[] args) throws IOException {
16        try {
17            socket = new Socket(host, port);
18            System.out.println("Client connected");
19            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
20            BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
21            String input;
22            while(true) {
23                System.out.print("Client sent: ");
24                input = stdIn.readLine();
25                out.write(input + '\n');
26                out.flush();
27                if(input.equals("bye"))
28                    break;
29            }
30        } catch (IOException e) {
31            System.out.println(e);
32        } finally {
33            if(socket!=null) {
34                socket.close();
35                System.out.println("Client socket closed");
36            }
37        }
38    }
```

Nội dung

- Giới thiệu
- Concurrent TCP Server
- Tạo Thread
- ThreadPool
- Excutor Framework
- Ví dụ ThreadPoolExecutor
- Ví dụ TCPSocket multithread
- Bài tập

Bài tập

- Viết chương trình server dùng TCPSocket, sử dụng multithread cài đặt các chức năng sau:
  - Tính toán: nhận từ client chuỗi phép tính gồm hai số và 1 phép tính → trả về kết quả tính toán. Cú pháp lệnh: calc 4+7
  - Phân tích số: nhận từ client 1 số n nguyên dương → trả về kết quả phân tích số thành tích các số nguyên tố. Cú pháp lệnh: prime 157

Tham khảo

- http://tutorials.jenkov.com/java-util-concurrent/executorservice.html
- https://viblo.asia/p/thread-pools-trong-java-ZK1ov1DxG5b9
- https://dzone.com/articles/the-executor-framework
- https://gpcoder.com/3548-huong-dan-tao-va-su-dung-threadpool-trong-java/