Lab9: 我超，并！

- </> 从进程说起
- 多线程
- ⚠ 多线程的问题

</> 从进程说起

- 代码是如何从文字变成计算机中的程序的?
- 多进程（多道）的实现，操作系统的演变

> 程序本身只是指令、数据及其组织形式的描述，相当于一个名词，行程才是程序（那些指令和数据）的真正执行实例，可以想像说是现在进行式。若干行程有可能与同一个程序相关系，且每个行程皆可以同步（循序）或异步（平行）的方式独立执行。现代计算机系统可在同一段时间内以进程的形式将多个程序加载到存储器中，并借由时间共享（或称时分复用），以在一个处理器上表现出同时（平行性）执行的感觉。

- 不足之处：创建进程很慢，进程通信不方便，阻塞等问题。

> 进程是系统分配资源的单位；线程是CPU调度和执行的单位。

多线程

- 线程——迷你进程
- 在**同一个进程**中同时执行多个线程，提高处理性能。

  > 这意味着共享内存地址空间

- 多线程也分为软件多线程和硬件多线程。

  > python多线程：臭名昭著

- 什么时候需要多线程：遇到绕不过去的弯。

  > 各种一个进程确确实实需要“同时”处理多件事务的情况，类似操作系统。例如最常见的
  >
  > - 网络服务器要同时服务多个用户
  > - 软件UI渲染与运算之间的冲突

多线程

Java如何实现多线程?

- 继承Thread类:

> 1. 定义某个类，使其继承Thread;
> 2. 重写Thread类中的run方法;
> 3. 调用线程的start方法。

```
public class ThreadDemo {
  public static void main(String[] args) {
        MyThread myThread = new MyThread("myThread");
        myThread.start();
        System.out.println("this is main.");
    }
}
class MyThread extends Thread{
    MyThread(String name){
        super(name);
    }
    public void run(){
    System.out.println("this is thread:"+this.getName());
    }
}
```

多线程

- 实现Runnable接口：

  1. 定义某个类，使其实现Runnable接口
  2. 重写Runnable中的run方法；
  3. 通过Thread类建立线程对象,将Runnable的子类对象作为参数传递给Thread类的构造函数；
  4. 调用Thread类的start方法启动线程并调用run方法。

```
public class RunnableDemo {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread thread0 = new Thread(myRunnable);
        thread0.start();
        System.out.println("this is main");
    }
}
class MyRunnable implements Runnable{
    private int count = 10;
    public void run() {
        while(count --> 0) System.out.println(Thread.currentThread().getName()+": counter:"+count);
    }
}
```

多线程

- 通过Callable和Future：

  1. 创建Callable接口的实现类，并实现call()方法，改方法将作为线程执行体，且具有返回值；
  2. 创建Callable实现类的实例，使用FutrueTask类进行包装Callable对象，FutureTask对象封装了Callable对象的call()方法的返回值；
  3. 使用FutureTask对象作为Thread对象启动新线程；
  4. 调用FutureTask对象的`get()`方法获取子线程执行结束后的返回值。

```java
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
public class CallableDemo {
    public static void main(String[] args) {
        MyCallable myCallable = new MyCallable();
        FutureTask<Integer> futureTask = new FutureTask<Integer>(myCallable);
        Thread thread = new Thread(futureTask, "Sub thread 0");
        thread.start();
        try{
            System.out.println("Sub thread returns: "+futureTask.get());
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }
}
class MyCallable implements Callable<Integer> {
    public Integer call() throws Exception {
        int count = 0;
        for(; count<100; count++) {
            System.out.println(Thread.currentThread().getName()+": count: "+ count);
        }
        return count;
    }
}
```

⚠ 多线程的问题

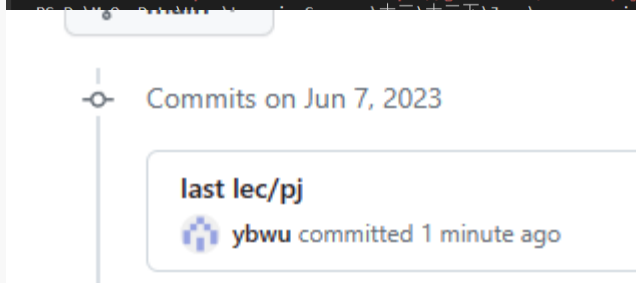
- 开销

  > CPU调度的开销是非常大的，尤其是当线程数过多的时候。

- 并发控制

  线程安全(MT-SAFE) 问题。线程同步，资源抢夺，以及死锁。

  ```
  Total 8 (delta 2), reused 0 (delta 0), pack-reused 0
  remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
  To https://github.com/ecnu-oop-java/ecnu-oop-java-up.git
   ! [remote rejected] main -> main (cannot lock ref 'refs/heads/main': is at 6490ad7b56800011d58835273023e6de954cc384 but expected eff69e61ad5488e7dcfacb036ad3d4f2c8fab423)
  error: failed to push some refs to 'https://github.com/ecnu-oop-java/ecnu-oop-java-up.git'
  ```

- 如何解决？加锁(`sychronized`)。

```
public class RunnableDemo {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        for(int i=0;i<1000;++i) new Thread(myRunnable).start();
        System.out.println(myRunnable.getCount());
    }
}

class MyRunnable implements Runnable{
    private int count = 0;
    public void run() {
        for(int i=0;i<1000;++i) count++;
    }
    public int getCount(){
        return this.count;
    }
}
```