The God We Trust

|  |
| --- |
|  |

7th Assignment Report

<محمدمهدی قلاوند> <۹۸۲۲۲۱۲۱>

**Introduction:**

1. در این تمرین قصد داریم موارد زیر را انجام دهیم

* در این تمرین مقدار عدد PI تا 1000 رقم اعشار بعد از ممیز به صورت شناور محاسبه کنیم
* سه Thread مختلف را مدیریت میکنیم و اطمینان حاصل میکنیم که ترتیب اجرای بین آنها به درستی انجام شود. برای انجام آن از CountDownLatch استفاده میکنیم
* مشکل همگام سازی را با استفاده از Semaphore حل میکنیم که به دو رشته اجازه میدهد تا وارد بخش بحرانی شوند

**Design and Implementation:**

برای اجرای همزمانی CountDownLatch یک کلاس مهم در جاوا میباشد که تضمین می کند یک یا چند Thread در صف قرار دارند تا Thread هی دیگر مجموعه عملیات خود را تکمیل کنند براساس مقدار شمارش CountDownLatch برای چندین منظور استفاده میشود. هنگامی که CountDownLatch را با مقدار شمارش 1 شروع می کنیم به سادگی به عنوان یک قفل روشن / خاموش عمل می کند وقتی CountDownLatch را با مقدار n شروع می کنیم باعث می شود که یک Thread منتظر بماند تا N Thread برخی اقدامات را انجام دهند یا تکمیل کنند

در جاوا از Semaphore در هنگام سازی Thread ها استفاده می شود.برای کنترل دسترسی به یک منبع مشترک از یک متغییر شمارنده استفاده می کند

**Testing and Evaluation:**

* طریقه پیاده سازی CountDownLatch

public class CountDownLatchDemo{

public static void main(String args[])

throws InterruptedException{

// Lets create a task that needs to wait for four threads before it begins

CountDownLatch latch = new CountDownLatch(4);

// Let's create four employee threads and begin them.

Employee first = new Employee(1000, latch,"EMPLOYEE-1");

Employee second = new Employee(2000, latch,"EMPLOYEE-2");

Employee third = new Employee(3000, latch,"EMPLOYEE-3");

Employee fourth = new Employee(4000, latch,"EMPLOYEE-4");

first.start();

second.start();

third.start();

fourth.start();

// The main task waits for four threads latch.await();

// Main thread has started

System.out.println(Thread.currentThread().getName() +" has finished");

}

}

// A class to represent threads for which the main thread waits.

class Employee extends Thread{

private int delay;

private CountDownLatch latch;

public Employee(int delay, CountDownLatch latch,String name){

super(name);

this.delay = delay;

this.latch = latch;

}

@Override

public void run(){

try{

Thread.sleep(delay);

latch.countDown();

System.out.println(Thread.currentThread().getName()+ " finished");

}

catch (InterruptedException e){

e.printStackTrace();

}

}

}

طریقه پیاده سازی Semaphore :

1. Semaphore sem= **new** Semaphore(1);
2. ...
3. sem.acquire();
4. **try**
5. {
6. //critical section
7. }
8. **finally**
9. {
10. sem.release();
11. }

**Conclusion:**

Thread کوچک‌ترین واحد اجرایی در سیستم عامل است. Thread‌ واحد اساسی اجرایی یا واحد اساسی در بهره‌برداری از CPU به حساب می‌آید. Thread یکی از مفاهیم مهم در سیستم‌های عامل محسوب می‌شود. Thread موجودیتی در درون یک پردازه (پروسه | Process) است که می‌تواند برای اجرا زمان‌بندی شود. یک برنامه تحت اجرا به عنوان یک Process شناخته می‌شود. هر برنامه ممکن است تعدادی پردازه مرتبط داشته باشد و هر پردازه می‌تواند دارای چند Thread باشد که این Threadها آن پردازه را اجرا می‌کنند. تمام Threadهای یک پردازه فضای آدرس مجازی و منابع سیستمی آن پردازه را با یکدیگر شریک می‌شوند..