KONKURENTNÉ PROGRAMOVANIE

8. cvičenie: Ukončovanie úloh, vlákien a exekútorov 2

Informovanie vlastníka o zrušení

- Ak máme kód úlohy, v ktorom môže nastať InterruptedException
  - Môžeme túto výnimku iba vyhodiť cez throws
    - Konečne je pohodlné riešenie to správne ☺
    - V Runnable implementáciách sa to nedá – kontrolovaná výnimka (v Callable to ide)
  - Alebo ju odchytíme a znova vyhodíme
  - Alebo ju odchytíme a nastavíme interrupted na true
    - Thread.currentThread().interrupt();
- InterruptedException nikdy nezhltávame!
  - Kód, ktorý nás spustil, by nemusel zistiť, že prebieha rušenie

Príklad konzumera

```
public Task getNextTask(BlockingQueue<Task> queue) {
  boolean zrušené= false;
  try {
    while (true) {
      try {
        return queue.take();
      } catch (InterruptedException e) {
        zrušené = true;
        // vlastník nás prerušuje - začneme nejaké upratovanie
      }
    }
  } finally {
    if (zrušené)
      Thread.currentThread().interrupt();
  }
}
```

Negatívny príklad rušenia vlákna, ktoré nevlastníme

```
private ScheduledExecutorService cancelExec = ...;
...
public void timedRun(Runnable r, long timeout, TimeUnit unit) {
  final Thread taskThread = Thread.currentThread();
  cancelExec.schedule(new Zrušiteľ(taskThread), timeout, unit);
  r.run();
}
```

```
public class Zrušiteľ implements Runnable {
  private Thread cudzieVlákno;
  public Zrušiteľ(Thread cudzieVlákno) {
     this.cudzieVlákno = cudzieVlákno;
  }
  public void run() {
     cudzieVlákno.interrupt();
  }
}
```

```
spustacUloh.timedRun(úloha, 1, TimeUnit.SECONDS)}
```

Negatívny príklad rušenia vlákna, ktoré nevlastníme

- Čo sa môže v tomto príklade stať?
  - Ak úloha, ktorú sme spustili, nezisťuje interrupted stav svojho vlákna,
    - vôbec to nemusí bežať iba 1 sekundu, ale pokojne do nekonečna
  - Ak úloha skončí skôr, ako nastane interrupt
    - Vlákno už môže robiť úplne inú úlohu, ktorú takto zrušíme
- **Nikdy neprerušujeme cudzie vlákna, ktoré sme nevytvárali!!!**

Rušenie úlohy cez Future

- návratová hodnota metódy submit() exekútorov
  - Future future = exekútor.submit(úloha)
- najpohodlnejší spôsob rušenia úlohy
  - future.**cancel**(boolean prerušiťVlákno)
    - Ak sa úloha ešte nezačala vykonávať, tak sa ani nevykoná
    - Ak sa začala vykonávať a prerušiťVlákno je false, úloha sa nechá dobehnúť
    - Ak sa začala vykonávať a prerušiťVlákno je true, **nastaví vláknu interrupted na true**

Riešenie časovanej úlohy cez Future

```
void timedRun(Runnable r, long timeout, TimeUnit unit) throws
                                                  InterruptedException {
  Future<?> future = exekútor.submit(r);
  try {
    future.get(timeout, unit);
  } catch (TimeoutException e) {
    // úloha ukončená po timeoute
  } catch (ExecutionException e) {
    // úloha vyhodila výnimku, vyhodíme ju tiež
    throw e.getCause();
  } finally {
    // nastaví interrupt, ak úloha ešte beží
    future.cancel(true);
  }
}
```

Ak vlákno niekto zrušil pred timeoutom

Ukončovanie exekútorov

- Ukončujeme všetky úlohy poslané do exekútora
- ExecutorService má metódy
  - void **shutdown()**
    - Exekútor neprijíma ďalšie úlohy
      - submit(úloha) vyhodí RejectedExecutionException
    - Všetky prijaté úlohy dovykonáva a potom skončí
  - List<Runnable> **shutdownNow()**
    - Exekútor neprijíma ďalšie úlohy
    - Úlohy, ktoré nezačali sú vrátené ako list Runnable úloh
    - Úlohám, ktoré bežia, sú prerušené ich vlákna cez interrupt()

Príklad - Logovacie vlákno

```
public class LogService {
  private final ExecutorService exec = newSingleThreadExecutor();
  ...
  public void stop() throws InterruptedException {
    try {
      exec.shutdown();
      exec.awaitTermination(TIMEOUT, UNIT);
    } finally {
      printWriter.close();
    }
  }

  public void log(String msg) {
    try {
      exec.execute(new WriteTask(msg));
    } catch (RejectedExecutionException ignored) { }
  }
}
```

Čo bolo prijaté bude zalogované, ak nevyprší timeout

Získanie úloh, ktoré boli prerušené

- Na to metódu nemáme, vieme to však nakódiť
  - Prerušené úlohy si zapamätáme v zdieľanej kolekcii

Zadanie 1

- Modifikujte riešenie zadania 2 z minulého cvičenia tak, aby:
  - sa po danom časovom intervale prestalo prehľadávať
  - okrem ukončených prehľadávaní, nech sa vypíšu aj čiastočne prehľadané podstromy aj s ich doteraz zistenou veľkosťou

Riešenie zadania 3

- DirAnalyzer testuje v každom adresári Thread.*currentThread().isInterrupted()*
  - Vyhodí výnimku DirAnalyzerException, v ktorej zahrnie adresár, jeho doteraz zistenú veľkosť a dôvod tejto výnimky
- Spustíme doterajší main ako úlohu v novom exekútore (viď slajd o časovanej úlohe cez Future)
  - Prečíta si dobehnuté úlohy a list prerušených úloh
  - Keď odchytí InterruptedException pošle svojmu exekútoru shutdownNow()
  - Volajúci kód rozlíši
    - future.isDone() – úloha dobehla ok
    - inak nezačala
    - Výnimka: úloha bola prerušená (buď nemáme oprávnenia, alebo sme boli prerušení)
  - Nakoniec nastaví Interrupted svojho vlákna a zatvorí svoj exekútor

Sumarizácia – rušenie úloh

- Rušenie úlohy v našich vláknach
  - vlákno.interrupt();
  - odchytenie výnimiek úlohy v hlavnom vlákne nemožné
- Rušenie jednej úlohy v thread-poole/Exekútore
  - future.cancel(true);
  - odchytenie výnimiek úlohy cez future.get() zabalenej v ExecutionException
- Rušenie všetkých úloh v exekútore
  - exekútor.shutdownNow();
  - odchytenie výnimiek úlohy cez future.get() zabalenej v ExecutionException

Sumarizácia – odhaľovanie zrušenia

- Dostatočne často overujeme Thread.currentThread().isInterrupted()
- Väčšina blokovaných operácií vyhadzuje InterruptedException
  - Rušia tým stav interrupted
  - Potom buď opätovne vyhodíme InterruptedException alebo nastavíme Thread.currentThread().interrupt()
- Na interrupted nie sú citlivé streamy a kritické sekcie spravené cez synchronized – dá sa to však obísť