به نام خدا

تمرین سری ششم

مائده بادان فیروز- ۹۸۲۲۲۰۰۹

بهار ۱۴۰۲

مقدمه:

هدف این تمرین آشنایی با multi thread و استفاده از آن است.

طراحی و پیاده سازی:

۱) برای بخش اول که شبیه‌سازی یک CPU تک هسته‌ای بود برای هر task یک thread ساختیم تا آن کار را انجام دهد. به وسیله‌ی یک حلقه‌ی for each، threadها ساخته شدند و در یک لیست آرایه ذخیره شدند. بعد در یک حلقه‌ی دیگر همگی start شدند و بعد از آن متد join برای آن‌ها فراخوانده شد تا main thread قبل از خاتمه یافتن باقی thread ها به پایان نرسد. اگر join استفاده نشود برنامه پیش از اتمام باقی thread ها تمام می‌شود!

۲) در بخش دوم برای انجام کار از pool thread استفاده کردیم. در هر برنامه ۴ thread ساخته می‌شود و اعداد به آن‌ها داده می شوند تا چک شوند که مضرب ۳ یا ۵ یا ۷ هستند یا نه. در این برنامه برای دسترسی به منبع مشترک sum، Race condition اتفاق می‌افتد که باید مدیریت شود. برای رفع این مشکل از lock استفاده کردیم. به این صورت که یک object از lock ساخته شد و به هر کلاسِ implement شده از Runnable داده شد تا در ناحیه بحرانی ابتدا قفل شود و بعد باز شود تا thread بعدی بتواند کارش را انجام دهد.

۳) در بخش سوم ابتدا با توجه به کدهای نوشته شده به راحتی با چک کردن دستی معلوم شد که هر دو thread بیش از ۳ ثانیه طول می‌کشند. اما برای اینکه در برنامه هم به طور دقیق و موثق چک شود یک thread کمکی ساختیم که با استفاده از آن گذر ۳ ثانیه را اندازه بگیریم (با sleep کردن آن به مدت ۳ ثانیه) و بعد چک کنیم که آیا هنوز thread مورد نظر زنده است یا نه. اگر زنده بود آن را با استفاده از متد interrupt، terminate می‌کنیم. همچنین کد کوتاهی در متد run خود threadها اضافه می‌کنیم که بعد از interrupt شدن متوجه شوند و کارشان را متوقف کنند.

سنجش و ارزیابی:

در سنجش بخش دوم امری عجیب بود که هرtest به تنهایی درست run می‌شد اما وقتی همه با هم اجرا می‌شدند جواب‌های اشتباه به دست می‌آمد! متوجه شدم که این به خاطر این است که بعد از هر بار محاسبه در تست برنامه متغیر sum از نو صفر نمی‌شود و با تست بعدی مقادیر جدید به مقدار ناصفر قبلی افزوده می شوند و از همین رو تست‌ها با شکست مواجه می‌شوند از این رو صرفا برای رفع این مشکل در خط ۵۶ کد زیر را

```
sum = 0;
```

اضافه کردم تا تست‌ها نیز درست باشند! طبیعی است که وجود این خط کد در برنامه ی اصلی غیرضروری و اضافی است.

در سنجش بخش سوم ابتدا دیدیم که پس از چاپ اینکه متد interrupt شده باز هم برنامه ادامه دارد و خاتمه نمی‌یابد! متوجه شدم که به خاطر این است که interrupt کردن یک thread صرفا به thread این آگاهی را می‌دهد که interrupt شده اما در رویه‌ی کار آن هیچ تغییری ایجاد نمی‌کند و خودمان باید با یک if چک کنیم که اگر interrupt شده باشد چکاری انجام بدهد. که بعد از آن با استفاده از متد ()interrupted. چک کردم که اگر interrupt شده بود از حلقه خارج شود و بدین ترتیب مشکل عدم پایان‌پذیری برنامه حل شد.

سنجش بخش دیگر با مشکل خاصی مواجه نشد.

نتیجه‌گیری:

Threadها برای سرعت بخشی به اجرای برنامه می‌توانند خیلی مفید واقع شوند اما این بهینگی هم حدی دارد و اگر از یک حدی بیشتر برای تقسیم کاری بین coreهای CPU، thread ایجاد کنیم دوباره بهینگی را از دست می‌دهیم و دچار زمان طولانی‌تر می‌شویم.

Race conditionها و هندل کردن آن‌ها در برنامه‌های multi thread خیلی مهم و حیاتی است. به طوری که اگر به اشتباه به حال خود رها شوند نتیجه‌ی کل برنامه غلط و غیرقابل استناد می‌شود.