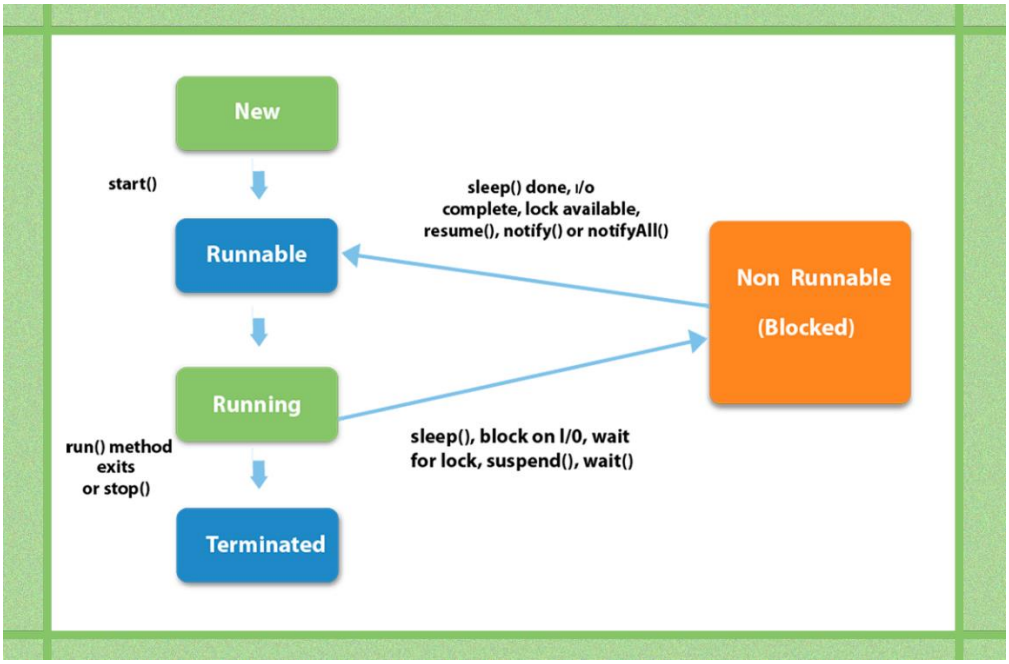
برنامه‌نویسی چندنخی در جاوا



برنامه‌نویسی چندنخی (Multithread Programming) به معنی اجرای همزمان (یا همروند) چند دنباله از دستورالعمل‌ها (نخ‌ها) است. استفاده از چند نخ (علاوه بر اینکه قابلیت‌های برنامه‌نویسی بیشتری در برخی کاربردها می‌دهد) امکان بهره‌گیری از اجرای موازی روی چندهسته پردازنده چندهسته‌ای و افزایش سرعت اجرا را نیز می‌دهد.

ساخت نخ در جاوا

قدم نخست برای استفاده از برنامه‌نویسی چندنخی در جاوا، ایجاد نخ است. ایجاد نخ در جاوا از دو راه زیر امکان پذیر است:

- ایجاد یک زیرکلاس از کلاس `java.lang.Thread`
- ایجاد کلاسی که واسط `java.lang.Runnable` را پیاده سازی کند.

در هر دو این روش‌ها متد `run()` باید در کلاس ایجاد شده پیاده‌سازی شود. متد `run()` در واقع شامل تمامی وظایفی است که نخ مورد نظر باید انجام دهد.

در روش نخست، بعد از تعریف کلاس جدیدی که از کلاس `Thread` ارث‌بری کرده و پیاده‌سازی متد `run()`، با ایجاد یک شی از کلاس ساخته شده و فراخوانی متد `start()` بر روی آن شی، مانند کد زیر به نخ ایجاد دستور شروع انجام وظایفش را می‌دهیم.

```
MyThread t = new MyThread();
t.start();
```

در روش دوم، پس از ایجاد کلاسی که واسط `Runnable` را پیاده‌سازی می‌کند و پیاده‌سازی متد `run()`،یک شی از کلاس `Thread` ایجاد می‌کنیم و به سازنده‌ی آن یک شی از کلاس جدیدی که ساخته‌ایم، پاس می‌دهیم. با فراخوانی متد `start()` بر روی شی نمونه کلاس `Thread`، مانند روش قبل اجرای نخ مورد نظر را شروع می‌کنیم.

```
Thread t = new Thread(new MyRunnable());
t.start();
```

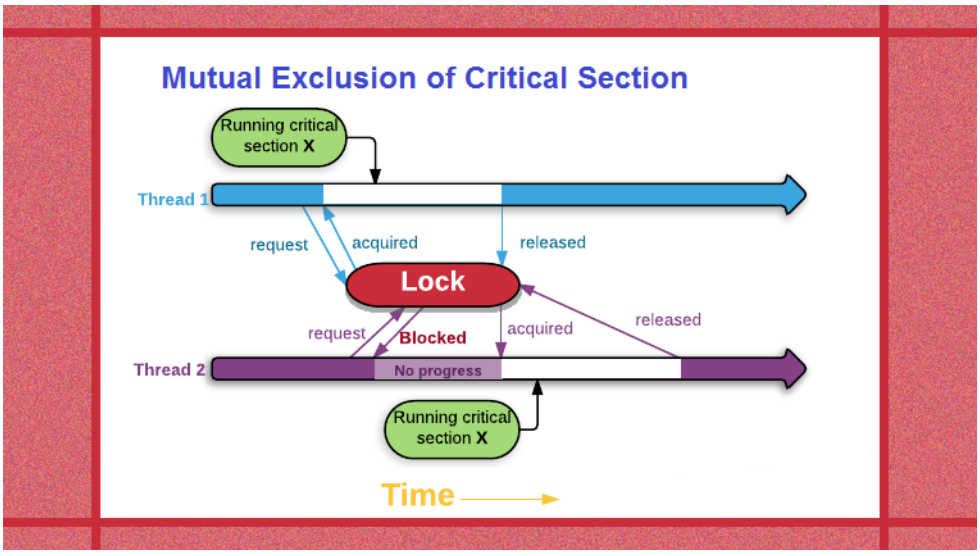
مدیریت نخ در ماشین مجازی جاوا

نخ‌ها را می‌توان به دو دسته نخ‌های بومی سیستم‌عامل و نخ‌های مجازی دسته‌بندی کرد. نخ‌های مجازی یا به اصطلاح Green Threads نخ‌هایی هستند که توسط کتابخانه زمان اجرا یا ماشین مجازی مدیریت و برنامه‌ریزی می‌شوند. نخ‌های مجازی همانطور که گفته شد برخلاف نخ‌های سیستم‌عامل، بجای حالت kernel در حالت user مدیریت می‌شوند. هدف کلی از ایجاد این نخ‌های مجازی، شبیه‌سازی و بهره بردن از مزایای یک محیط چند نخی در ابزارهایی است که توانایی استفاده از امکانات بومی سیستم‌عامل‌ها و نخ‌های آن‌ها را ندارند. اصطلاح Green Threads در واقع اشاره به نام اصلی کتابخانه نخ‌ها در زبان برنامه‌نویسی جاوا دارد.

تا نسخه ۱.۱ ماشین مجازی جاوا، Green Threads تنها مدل چندنخی بود اما به مرور در نسخه‌های بعدی به دلیل برتری نخ‌های بومی سیستم‌عامل به نخ‌های مجازی، استفاده از آن‌ها محدودتر شد و اکنون پشتیبانی از این نخ‌ها صورت نمی‌گیرد. با این وجود، بنا به دلایلی مانند کمینه کردن استفاده از native code و سبک‌تر بودن این نخ‌ها نسبت به نخ‌های سیستم‌عامل، هنوز هم از این نخ‌های مجازی در برخی موارد استفاده می‌شود؛ برای مثال می‌توان به Squawk Virtual Machine اشاره کرد.

نواحی بحرانی و بلاک‌های همگام‌سازی

نواحی بحرانی بخش‌هایی از کد هستند که اگر همزمان توسط دو نخ اجرا شوند، ممکن است مشکلاتی را در نتایج خروجی کد ایجاد کند (برای توضیحات بیشتر در خصوص شرایط مسابقه و ناحیه بحرانی می‌تواند اینجا را بخوانید).



در زبان برنامه نویسی جاوا قسمت‌های بحرانی با کلیدواژه synchronized مشخص می‌شوند. نخ‌ها برای ورود به نواحی بحرانی نیاز دارند تا قفلی که برای آن ناحیه بحرانی در نظر گرفته شده است را در اختیار بگیرند و پس از خروجی از ناحیه‌ی بحرانی قفل در اختیار گرفته شده را رها کنند.

در زبان برنامه نویسی جاوا هر شی خود یک قفل نیز دارد یعنی می‌توان از هر شی‌ای به عنوان قفل استفاده کرد. به طور مثال اگر یک متد به صورت `synchronized` پیاده‌سازی شود و بر روی یک شی فراخوانی شود، قفل همان شی را در اختیار می‌گیرد. علاوه بر مطالب گفته شده نواحی بحرانی می‌توانند از قفل‌های دیگری نیز استفاده کنند؛ مانند کد زیر که از .قفل شی `names` استفاده می‌کند.

```
List<String> names;
...
synchronized(names){
  names.add("ali");
{
```

توجه کنید متدهایی که به صورت همزمان به صورت `static` و `synchronized` تعریف می‌شوند، نمی‌توانند به صورت همزمان بر روی دو یا چند شی از یک کلاس یکسان فراخوانی شوند. این متدها به جای **قفل شی** که بر روی آن فراخوانی می‌شوند، **قفل کلاس** آن شی را به دلیل `static` بودن در اختیاز می‌گیرند.

در نهایت می‌توان گفت زبان برنامه‌نویسی جاوا برای مدیریت شرایط مسابقه و نواحی بحرانی از بلاک‌های `synchronized` و قفل اشیا استفاده می‌کند.

ارتباط بین نخ‌ها و سازوکارهای همگام‌سازی

تا اینجا در خصوص نحوه ساخت نخ‌ها و مدیریت آن‌ها توسط ماشین مجازی جاوا صحبت کردیم. در بسیاری از برنامه‌های چندنخی اما علاوه بر ساختن نخ‌ها و مدیریت زمان شروع و پایان کار آن‌ها، نیاز به ارتباط بین خود نخ‌ها داریم. به عنوان مثال فرض کنید در یک برنامه برای انجام یک سری محاسبات، یک نخ داده‌های مورد نیاز برای هر مرحله از محاسبات را از طریق شبکه دریافت می‌کند و نخ دیگری بعد از رسیدن بخشی از داده‌های جدید، محاسبات را شروع می‌کند. در چنین برنامه‌ای نخ دریافت‌کننده داده‌ها باید به نوعی رسیدن داده‌ها را به اطلاع نخ محاسبه‌گر برساند.

در این بخش به برخی از قابلیت‌هایی که در جاوا به منظور ارتباط بین نخ‌ها تعبیه شده است می‌پردازیم.

سازوکارهای سطح پایین ارتباط بین نخ‌ها

همانطور که می‌دانیم تمامی اشیا در جاوا به طور مستقیم یا غیرمستقیم از کلاس `Object` ارث‌بری می‌کنند. در نسخه‌های اولیه جاوا (پیش از نسخه ۵) برای ارتباط بین نخ‌ها از دو تابع `wait()` و `notify()` که در کلاس `Object` تعریف شده‌اند استفاده می‌شد. در این روش نخ‌ها با استفاده از یک یا چند شی مشترک، می‌توانستند به ارتباط با یکدیگر و همگام‌سازی بپردازند.

فرض کنید شی `obj` بین دو نخ `th1` و `th2` به اشتراک گذاشته شده است. زمانی که یکی از نخ‌ها (مثلا `th1`) روی `obj` متد `wait()` را صدا بزند، اجرای این نخ متوقف می‌شود و این نخ به حالت خواب (sleep) می‌رود. پس از این اگر نخ دیگری (مثلا `th2`) روی همان شی مشترک `obj` متد `notify()` را صدا بزند، یکی از نخ‌هایی که پیشتر روی `obj` عمل `wait` را انجام می‌دادند (در اینجا تنها نخ `th1` منتظر مانده است) توسط ماشین مجازی جاوا انتخاب می‌شود و اجرای آن ادامه پیدا می‌کند. به این صورت نخ‌ها می‌توانند با یکدیگر تعامل کنند و کارهای خود را با یکدیگر همگام کنند.

توجه کنید که صدا زدن دو متد `wait()` و `notify()` می‌تواند باعث ایجاد شرایط مسابقه‌ای شود و به همین دلیل جاوا شما را ملزم به استفاده از آن‌ها در بلاک `synchronized` همان شی می‌کند.

```
synchronized (obj) {
  obj.wait();
}
```
th1

```
synchronized (obj) {
  obj.notify();
}
```
th2

سازوکارهای سطح بالای همگام‌سازی نخ‌ها

از نسخه ۵ جاوا در بسته `java.util.concurrent` ابزارهای سطح بالاتر بسیاری به جاوا اضافه شد. استفاده از این ابزارها علاوه بر ساده‌تر کردن برنامه‌نویسی، می‌تواند کارایی بالاتری نسبت به ابزارهای سطح پایین نیز داشته باشد. ابزارهای این بسته را می‌توان به دو دسته ظرف‌های امن (thread-safe) و اشیای همگام‌ساز (synchronizer objects) تقسیم کرد.

ظرف‌های امن، ظرف‌هایی مانند آرایه، پشته و صف هستند که می‌توان بدون داشتن دغدغه رخ دادن شرایط مسابقه از آن‌ها استفاده کرد و این ظرف‌ها رخ ندادن شرایط مسابقه را مدیریت می‌کنند.

اشیای همگام‌ساز، اشیایی هستند که با به اشتراک‌گذاری یک نمونه از آن‌ها بین دو یا چند نخ، می‌توان با استفاده از امکاناتی که هر کدام در اختیار ما می‌گذارند، بین نخ‌ها همگام‌سازی (و در برخی تبادل) پیام انجام داد. در ادامه به توضیح مختصری در مورد چند شی همگام‌ساز پر استفاده در جاوا می‌پردازیم:

- سمافور (`Semaphore`): سمافور یک همگام‌ساز است که می‌توان با آن تعداد نخ‌هایی که همزمان از یک منبع مشترک استفاده می‌کنند را کنترل کرد. سمافور یک حالت داخلی دارد که تعداد نخ‌هایی که اجازه ورود به یک ناحیه را دارند نگه می‌دارد. با هر بار صدا زدن `acquire()` روی سمافور، این عدد یک واحد کم می‌شود و در صورت رسیدن به صفر، اجازه ورود نخ دیگری را به آن ناحیه نمی‌دهد. با صدا زدن `release()` روی سمافور نیز این عدد یک واحد افزایش می‌یابد و اگر نخ دیگری روی سمافور منتظر است، اجرای آن ادامه پیدا می‌کند.
- `CountDownLatch`: این شی یک شمارنده داخلی دارد که با هر بار صدا زدن `countdown()` روی آن، یک واحد از عدد شمارنده کاسته می‌شود. در صورتی که پیش از رسیدن این عدد به صفر، یک یا چند نخ دیگر روی آن متد `await()` را صدا بزنند، اجرای آن‌ها تا زمانی که شمارنده صفر نشده است متوقف می‌شود.
- `Exchanger`: این کلاس می‌تواند بین دو نخ عمل همگام‌سازی و انتقال پیام را انجام دهد. در واقع دو نخ با صدا زدن `exchange()` روی این شی، علاوه بر اینکه اجرای خود را در یک نقطه همگام می‌کنند، یک شی را نیز می‌توانند بین خود جابجا کنند.
- `CyclicBarrier`: این شی شبیه به `Exchanger`، می‌تواند نخ یک نقطه تلاقی بین نخ‌ها ایجاد کند. با این تفاوت که بین بیش از دو نخ این کار امکان پذیر است. اما مانند `Exchanger` امکان تبادل پیام را ندارد.

استفاده از اشیای معرفی شده و بسیاری از اشیای دیگر که در این بسته وجود دارند، می‌تواند برنامه‌نویسی چندنخی را تا حد زیادی برای برنامه‌نویس ساده‌تر کند و استفاده از این ابزارها بسیار توصیه می‌شود.

نگارش: رادین شایانفر (۹۷۳۱۰۳۲) – محمدعرفان قاسمی (۹۷۳۱۰۴۸)

هر دو نگارنده این گزارش رضایت خود را در خصوص انتشار آن در سایت ابررایانه امیرکبیر اعلام می‌دارند.