التطبيقات الموزعة – الجلستين 4+5 – النياسب

أهداف الجلستين

ضمن الجلستين 4 و 5 سيتم التعرف على النياسب في لغة البرمجة جافا.

من المتوقع في نهاية الجلستين أن يكون الطالب قادر على:

- إنشاء و تشغيل النياسب في جافا.
- استخدم synchronized لمزامنة تنفيذ المقطع الحرج.
- مشاركة البيانات بين أكثر من نيسب باستخدام Blocking Queues

النياسب

يعتبر النيسب إجرائية خفيفة, حيث يمكن إنشاء أكثر من نيسب ضمن الإجرائية الواحدة و هذه النياسب تتشارك فيما بينها البيانات و الموارد مع الإجرائية الأم و يكون الكود البرمجي الخاص بالنيسب هو جزء من الكود البرمجي الخاص بالإجرائية.

استخدام النياسب يساهم في تحقيق الإستخدام الأمثل للأنوية المتعددة في المعالج حيث يتم تنفيذ كل نيسب يتم إنشاؤه على نواة مختلفة في حال كان هناك عدد كافي من الأنوية للنياسب, كما تسهل النياسب مشاركة الموارد و البيانات فيما بينها لأنها ترث من الإجرائية الأم الذاكرة المتاحة لها.

من أهم مساوئ النياسب هي أنها يمكن أن تسبب توقف تام إذا لم نقم بمزامنة الوصول و استخدام الموارد المشتركة فيما بينها.

أيضاً إنشاء عدد كبير من النياسب يستهلك وقت من وحدة المعالجة و أيضاً قد يكون لديه تأثيرات سلبية و خصوصاً في حال كان العمل الذي يقوم به النيسب CPU Intensive على عكس النياسب التي تقوم بأعمال IO Intensive, حيث يمكن إنشاء أعداد أكبر منها دون أي تأثيرات سلبية على الأداء.

النياسب في لغة البرمجة جافا

يتم إنشاء النياسب في جافا باستخدام طريقتين:

1. الوراثة من الصف Thread يتم إنشاء صف جديد يرث من الصف Thread و نعرف ضمنه الطريقة public void run التي تتضمن الكود البرمجي الخاص بالنيسب, نأخذ كائن من الصف الجديد و نستدعي الطريقة start عليه التي ترسل النيسب لمجدول نظام التشغيل كي يتم تنفيذه عندما يأتي دوره (الطريقة start لا تبدأ تنفيذ النيسب).
2. تحقيق الواجهة Runnable, نعرف صف جديد يحقق هذه الواهة و نعرف ضمنه الطريقة public void run التي تتضمن الكود الخاص بالنيسب, نأخذ كائن من الصف الجديد, و نأخذ كائن من الصف Thread و نمرر له ضمن الباني الكائن من الصف الخاص بنا ثم نستدعي الطريقة start على الكائن من الصف Thread.

يمكن استخدام أي طريقة من الطرق السابقة, نحتاج أن نستخدم الطريقة الثانية عندما يكون بدينا صف يرث من صف ما و في نفس الوقت نريد أن نجعله نيسب.

عند استخدام الطريقة الثانية يجب أن نستخدم الطريقة Thread.currentThread لنحصل على مؤشر للنيسب الحالي و نستدعي عليه الطرائق التي نريدها.

أهم الطراءق المعرفة على النياسب

الطريقة start ترسل النيسب للمجدول, الطريق getName تعيد اسم النيسب, الطريقة setName تحدد اسم النيسب, الطريقة join تجعل النيسب الحالي (الذي استدعائها) ينتظر النيسب الذي تم إستدعاء الطريقة عليه حتى ينتهي و بعدها يستطيع النيسب الحالي ان يكمل التنفيذ و الطريقة sleep تجع النيسب الحالي ينتظر عدد معين من milliseconds قبل أن يتابع التنفيذ.

الطريقة wait تجعل النيسب الحالي ينتظر حتى يقوم نيسب آخر بإيقاظه باستخدام الطريقة notify أو حتى يقوم نيسب آخر باستدعاء الطريقة notifyAll.

التزامن

عندما يتم مشاركة المتحولات بين عدة نياسب لا بد من تنسيق عمليات الوصول إليها و خصوصاً عند القيام بعملية الكتابة.

لنأخذ المثال البسيط التالي:

ليكن لدينا التعليمة التالية

```
int i = 0;
i = i + 1;
```

هذه العملية البسيطة التي تتضمن إضافة واحد للمتحول i تقسم لثلاث عمليات فرعية هي:

جلب قيمة i من الذاكرة و تخزينها ضمن أحد مسجلات المعالج, غضافة واحد للقيمة المخزنة و كتابة القيمة الجديدة ضمن الذاكرة, لنفترض أنه لدينا نيسبين للقيام بهذه العملية و جاء النيسب الأول لينفذ العملية بدأ بجلب قيمة i من الذاكرة و بعد هذه العملية مباشرة انتهت حصته الزمنية و تم إيقافه من العمل و جاء نيسب آخر ليقوم بعمله, فقام النيسب الجديد بتنفيذ العمليات الثلاث كاملة مخزناً القيمة 1 في الذاكرة, عندما يعود النيسب الأول للتنفيذ فإنه يتابع من العملية التي توقف عندها أي يزادة واحد للقيمة المخزنة ضمن مسجلات المعالج فتصبح 1 (عندما جلب قيمة i كانت صفر) و يقوم بتخزين القيمة الجديدة ضمن الذاكرة و التي هي واحد و بالتالي يحدث هنا الخطأ حيث قام نيسبين بزيادة قيمة i لكنها أصبحت واحد بدلاً من اثنين, لذلك لا بد من مزامنة عمل النيساب في حال الكتابة لموقع ذاكرة مشترك لتجنب هذه المشكلة, لتحقيق التزامن نستخدم synchronized كما يلي.

استخدام synchronized لمزامنة النياسب

يمكن استحدام العبارة synchronized لتحديد جزء من الكود على أنه مقطع حرج و بالتالي يمكن تنفيذه من قبل نيسب واحد فقط في نفس اللحظة و لا يمكن أن يقوم أكثر من نيسب بتنفيذه, أي عندما يبدأ نيسب ما تنفيذه و انتهت حصته قبل الإنتهاء منه لا يستطيع أي نيسب آخر تنفيذه قبل أن ينهي النيسب الأول عمله بالكامل.

كل استخدام للعبارة synchronized يتضمن مع كائن مراقب Monitor Object أي هذا الكائن يملك القفل لهذا المقطع و عندما يأخذ أحد النياسب هذا القفل لا يستطيع أي نيسب آخر أخذ القفل من نفس الكائن حتى يعيده النيسب الأول له, يمكن أن يكون هذا الكائن أي كائن في جافا مهما يكن.

يمكن استخدام synchronized بطريقتين:

- ضمن تعريف الطريقة و بالتالي يصبح الكائن المراقب هو الكائن من الصف الذي تنتمي الطريقة له و إذا كانت الطريقة static يتم استخدام الكائن الخاص بالصف الذي تنشأ الجافا نسخة واحدة منه لكل صف.
- ضمن كتلة خاصة بها و هنا نستطيع تمرير الكائن المراقب بين قوسين () و في حال كانت الكتلة موجود ضمن طريقة static نستطيع استخدام الكائن الخاص بالصف كما يلي ClassName.class.

القوائم في جافا

سنتعرف الآن على القوائم في لغة جافا بسبب الحاجة لها ضمن الفقرة التالية.

تعرف القائمة في جافا بأنها مصفوفة ديناميكية يمكن أن يزداد أو ينقص حجمها أثناء تنفيذ البرنامج و بشكل ديناميكي من دون أي حد سوى حجم الذاكرة المتاحة, هناك عدة أنواع للقوائم سنتعرف هنا على القوائم المترابطة فقط.

يتم تعريف قائمة مترابطة في جافا كما يلي:

```
LinkedList list = new LinkedList();
```

تعرف القوائم المترابطة الطرائق التالية.

الطريقة add لإضافة عنصر جديد في نهاية القائمة, الطريقة remove تحذف عنصر حسب دليله و ترد العنصر الذي تم حذفه, الطريقة size ترد عدد العناصر في القائمة, الطريقة get ترد عنصر حسب دليله, الطريقة set تغير عنصر حسب دليله حيث تمرر أولاً دليل العنصر المراد تغييره و ثانياً العنصر الذي نريد إضافته مكانه, الطريقة contains ترد true أو false حسب إذا كان العنصر موجود ضمن القائمة أو لاء, الطريقة indexOf ترد دليل العنصر الممر لها في حال كان غير موجود ترد -1.

يمكن الوصول لعناصر القئامة باستخدام حلقة for كما يلي:

```
for (Object item : list) {System.out.println(item);}
```

تم استخدام الصف Object هنا لأن عناصر القائمة يمكن أم تكون من أي صف و بالتالي بما أن الجميع يرث من Object يمكن استخدامه بهذا الشكل.

الرتل Blocking Queue

يتم إستخدام الرتل Blocking Queue لمشاركة البيانات بين أكثر من نيسب حيث يقوم نيسب أو مجموعة من النياسب بإضافة بيانات إليه و في حال كان الرتل ممتلئ يتم إيقاف النيسب و منعه من المتابعة حتى يقوم نيسب آخر بحذف بيانات منه و إيقاظ جميع النياسب التي تنتظر إضافة عنصر للرتل, و في الحالة الأخرى عندما يحاول نيسب حذف عنصر من رتل فارغ فإنه يتوقف حتى يقوم نيسب آخر بإدخال عنصر إليه.

يتم تحقيق هذا الرتل باستخدام القوائم المترابطة كما يلي:

```
class BlockingQueue {
	private LinkedList queue;
	private int limit;
	public BlockingQueue(int l) {
		limit = l;
		queue = new LinkedList();
	}
	public synchronized void add(Object item) {
		while (queue.size() == limit) {
			wait();
		}
		if (queue.size() == 0) {
			notifyAll();
		}
		queue.add(item);
	}
	public synchronized Object pop() {
```

```
            while (queue.size() == 0) {
                wait();
            }
            if (queue.size() == limit){
                notifyAll();
            }
            return queue.remove(0);
        }
}
```

حل التمارين

حلول التمرينين ضمن المحاضرة موجودة على github ضمن الرابط التالي.

https://github.com/mohsenSy/JavaThreads

Contents

أهداف الجلستين 1
النياسب 1
النياسب في لغة البرمجة جافا 1
أهم الطراءق المعرفة على النياسب 1
التزامن 2
لمزامنة النياسب synchronized استخدام 2
القوائم في جافا 2
الرتل Blocking Queue 3
حل التمارين 4