# Real Time Systems

## Thread Priorities

Dr. Marwa Dahdouh

# Scheduler (الجدولة)

- يتحكم مجدول VM في ال thread الذي يعمل داخل VM في أي وقت. هذا يعني أنه يحمي بشكل فعّال هياكل بيانات VM الداخلية مثل heap (الكومة) من التعديلات المتزامنة.
- تُظهر إعادة الجدولة النقطة التي يتغير فيها ال thread من running (يعمل) إلى ready (جاهز) بسبب طلب إعادة الجدولة.
- يوفر مجدول VM أولوية في الوقت الفعلي (real-time priority) داخل برامج Java على أنظمة التشغيل التي لا تقدم جدولة قائمة على الأولوية الصارمة.
- تقلل الجدولة من الحمل الزائد (overhead) لاستدعاءات JNI ويساعد نظام التشغيل على جدولة موارد وحدة المعالجة المركزية بشكل أفضل لل(threads) المرتبطة بـ VM.
- مجدول الأولويات (**PriorityScheduler**) --- طرق جديدة موروثة من المجدول:

javax.realtime.Scheduler
**Subclasses**:PriorityScheduler

# خصائص الأشياء القابلة للجدولة:

الكائن القابل للجدولة ( **schedulable object**) هو أي كائن ينفذ (implement) واجهة الجدولة ( Schedulable interface) (على الرغم من أن جدولة الأولويات تدعم فقط real-time threads ومعالجات الأحداث غير المتزامنة(asynchronous event handlers)).

- ❑ **بارامترات الإصدار**(**ReleaseParameters**) - إعطاء تكلفة المعالجة لكل إصدار للكائن وموعده النهائي ؛ إذا تم إطلاق ال object بشكل دوري أو متقطع، فإن الفئات الفرعية (subclasses) تسمح بإعطاء فترة زمنية.
- ❑ **بارامترات الجدولة** (**SchedulingParameters**) - فئة SchedulingParameters فارغة ؛ غير أن الفئات الفرعية تسمح بتحديد أولوية الشيء إلى جانب أهميته بالنسبة للأداء العام للطلب.
- ❑ **بارامترات الذاكرة** (**MemoryParameters**) - تعطي:
  - الحد الأقصى من الذاكرة التي يستخدمها ال object في منطقة ذاكرته الافتراضية.
  - الحد الأقصى من الذاكرة المستخدمة في الذاكرة الخالدة (Immortal Memory).
  - الحد الأقصى لمعدل تخصيص ذاكرة الكومة (Heap Memory).
- ❑ بارامترات مجموعة المعالجة (**ProcessingGroupParameters**) - يسمح هذا بمعاملة العديد من schedulable objects كمجموعة والحصول على فترة وتكلفة وموعد نهائي مرتبط بها.

# How to set SchedulingParameters for thread with priority

```
// create Schedulable sched;
Schedulable sched;
 private Schedulable sched = null;
 if ( sched instanceof RealtimeThread)
            ((RealtimeThread)sched).schedulePeriodic();
........
sched.setSchedulingParameters(pri);
sched.setReleaseParameters(per);
sched.start();
```

# أولويات ال Thread (*Thread Priorities*):

- تشترك كائنات **RealtimeThread** و **NoHeapRealtimeThread** و **Thread** العادية في نفس نطاق الأولوية.
- أقل أولوية ممكنة لل thread من أجل كل هذه ال Threads هي **MIN_PRIORITY** والتي يتم تحديدها في حزمة **java.lang**، class Thread.
- يمكن الحصول على أعلى أولوية ممكنة من خلال الاستفسار (querying):

**instance().getMaxPriority()** in package javax.realtime.x;

- تحديد الأولوية إلى أعلى:

```
int pri = PriorityScheduler.instance().getMaxPriority();
PriorityParameters sched = new PriorityParameters(pri);
```

- تحديد الأولوية إلى أدنى:

```
int pri = PriorityScheduler.instance().getMinPriority();
PriorityParameters sched = new PriorityParameters(pri);
```

- تحديد الأولوية كقيمة:

```
PriorityParameters sched = new PriorityParameters(int num);
this.setSchedulingParameters(sched);
```

# Methods

**Scheduler**() ينشئ instance من Scheduler.

**getDefaultScheduler**() يحصل على reference إلى ال scheduler الافتراضي.

**PriorityScheduler**() يقوم ببناء instance من PriorityScheduler.

**getMaxPriority**() الحصول على أقصى أولوية متاحة ل schedulable object يديره هذا المجدول.

**getMaxPriority**(java.lang.Thread thread)

يحصل على أعلى أولوية لل thread المعطى.

**getMinPriority**() الحصول على الحد الأدنى من الأولوية المتاحة ل schedulable object يديره هذا المجدول.

**getNormPriority**() الحصول على الأولوية العادية المتاحة ل schedulable object يديره هذا المجدول.

**PriorityScheduler instance**()

يعيد reference إلى ال instance المميز ل PriorityScheduler الذي هو الجدولة الأساسية للنظام.

# Wait-free communication

المزامنة (Synchronization):

- تُمكّن فئات WaitFreeReadQueue وWaitFreeWriteQueue وWaitFreeDequeue التواصل بدون انتظار بين كائنات قابلة للجدولة(schedulable objects) (خصوصًا instances of NoHeapRealtimeThread) و ثريدات جافا العادية.

- توفر فئات الصفوف بدون انتظار (The wait-free queue classes ) تحديد الوصول المتزامن والآمن للبيانات المشتركة بين instances of NoHeapRealtimeThread وكائنات قابلة للجدولة تخضع لتأخير جمع النفايات (garbage).

# Class WaitFreeWriteQueue

- **javax.realtime.WaitFreeWriteQueue.** public class **WaitFreeWriteQueue** extends java.lang.Object

- تعد فئة WaitFreeWriteQueue رتلاً (queue) يمكن أن يكون غير محجوب للمنتجين (producers). تم تصميم فئة WaitFreeWriteQueue للتواصل بين كاتب واحد(single-writer) وقارئين متعددين(multiple-reader)، على الرغم من أنه يمكن استخدامها أيضًا (بحذر) مع كتّاب متعددين(multiple writers).
- الكاتب (writer) عادةً ما يكون instance of NoHeapRealtimeThread، بينما القراء (readers) عادةً ما يكونون java threads العادية أو heap-using real-time threads أو كائنات قابلة للجدولة(schedulable objects).
- يتم التواصل من خلال ذاكرة مؤقتة محدودة من الكائنات(bounded buffer of Objects) تُدار بنظام first-in-first-out.
- طريقة write تقوم بإضافة عنصر جديد إلى نهاية القائمة. هذه الطريقة غير متزامنة ولا تحظر عندما تكون القائمة ممتلئة (تُرجع قيمة false بدلاً من ذلك).
- يُسمح بوجود عدة ثريدات كتابة (writer threads) أو كائنات قابلة للجدولة، ولكن إذا كان هناك اثنان أو أكثر من الثريدات التي ترغب في الكتابة إلى نفس WaitFreeWriteQueue، سيحتاجون إلى تنظيم تزامن صريح.
- طريقة read تقوم بإزالة أقدم عنصر من الرتل. هذه الطريقة متزامنة وتقوم بالحظر عندما تكون القائمة فارغة.

- قد يتم استدعاءها من قبل أكثر من قارئ وفي هذه الحالة، سيقرأ المستدعون المكونات المختلفة من الرتل.

# Class WaitFreeWriteQueue

**WaitFreeWriteQueue** هي واحدة من الفئات التي تسمح لـ **NoHeapRealtimeThreads** وثريدات جافا العادية (**regular java threads**) بالمزامنة على كائن من دون خطر تكبد **NoHeapRealtimeThread** تأخير جامع النفايات (Garbage Collector ) بسبب إدارة تجنب تقلب الأولوية(priority inversion avoidance management).

تقوم بإنشاء queue في الذاكرة بطريقة () write غير متزامنة وغير قابلة للحجب، وطريقة () read متزامنة وقابلة للحجب.

**البارامترات:** **writer** - instance من thread أو كائن قابل للجدولة (schedulable object)، أو null.

**reader** - instance من thread أو كائن قابل للجدولة (schedulable object)، أو null.

**maximum** - الحد الأقصى لعدد العناصر في الرتل.

**memory** - منطقة الذاكرة التي يتم فيها تخصيص هذا الكائن والعناصر الداخلية له.

## Constructor Summary

| |
|---|
| `WaitFreeWriteQueue(int maximum)`<br>ينشئ رتل يحتوي على أقصى عدد من العناصر في الذاكرة الخالدة (Immortal Memory). |
| `WaitFreeWriteQueue(int maximum, MemoryArea memory)`<br>ينشئ رتل يحتوي على أقصى عدد من العناصر في الذاكرة. |
| `WaitFreeWriteQueue(java.lang.Runnable writer, java.lang.Runnable reader, int maximum, MemoryArea memory)`<br>تقوم بإنشاء queue في الذاكرة بطريقة () write غير متزامنة وغير قابلة للحجب، وطريقة () read متزامنة وقابلة للحجب. |

# Class WaitFreeWriteQueue

| Method Summary | |
|---|---|
| void | **clear**()<br>يقوم بتعيين الرتل إلى حالة فارغة. |
| boolean | **force**(java.lang.Object object)<br>يقوم بإدراج الكائن دون قيد أو شرط في الرتل ، إما في موقع شاغر أو الكتابة فوق أحدث عنصر تم إدراجه. |
| boolean | **isEmpty**()<br>يستعلم النظام لتحديد ما إذا كان الرتل فارغ أم لا. |
| boolean | **isFull**()<br>يستعلم النظام لتحديد ما إذا كان الرتل ممتلئ أم لا. |
| java.lang.Object | **read**()<br>عملية متزامنة ومن المحتمل حظرها في قائمة الانتظارqueue. |
| int | **size**()<br>يستعلم عن قائمة الانتظار لتحديد عدد العناصر في الرتل. |
| boolean | **write**(java.lang.Object object)<br>يقوم بإدراج الكائن في الرتل إذا كان الرتل غير ممتلئ، وإلا فليس له أي تأثير على الرتل؛ وتُعكس قيمة ال boolean ما إذا كان قد تم إدخال الكائن أم لا. |

# Class WaitFreeReadQueue

- javax.realtime.WaitFreeReadQueue , public class **WaitFreeReadQueue** extends java.lang.Object.

- قائمة الانتظار queue يمكن أن تكون غير محظورة للمستهلكين consumers. تم تصميم فئة WaitFreeReadQueue للتواصل مع قارئ واحد متعدد الكتّاب ، على الرغم من أنه يمكن استخدامها أيضًا (بحذر) لقرّاء متعددين .

- يعد القارئ (**reader**) عمومًا instance لـ NoHeapRealtimeThread ، ويكون الكتّاب (**writers**) عمومًا عبارة عن regular Java threads أو heap-using real-time threads أو كائنات قابلة للجدولة schedulable objects .

- يتم الاتصال من خلال مخزن مؤقت محدود للكائنات (bounded buffer of Objects) يتم إدارته بطريقة **first-in-first-out** .
- تقوم طريقة الكتابة **write** بإلحاق عنصر جديد بقائمة الانتظار queue. تتم مزامنته ، ويتم حظره عند امتلاء قائمة الانتظار. قد يتم استدعاؤه من قبل أكثر من كاتب واحد writer، وفي هذه الحالة ، المستدعون المختلفون سيكتبون إلى عناصر مختلفة من قائمة الانتظار.
- تزيل طريقة القراءة read أقدم عنصر من قائمة الانتظار queue. لا تتم مزامنتها ولا تقوم بحظرها؛ سيعيد null عندما تكون قائمة الانتظار فارغة .
- يُسمح بِـ **Multiple reader threads** أو **schedulable objects** ، ولكن إذا كان اثنان أو أكثر ينوون القراءة من نفس WaitFreeWriteQueue، فسيحتاجون إلى ترتيب مزامنة صريحة .

# WaitFreeReadQueue

تنشئ قائمة انتظار queue تحتوي على أقصى عدد من العناصر في الذاكرة. تحتوي قائمة الانتظار على read() method غير متزامنة وغير قابلة للحظر و write() method متزامنة وقابلة للحظر .

**البارامترات:**

**maximum** - الحد الأقصى لعدد العناصر في قائمة الانتظار.

**notify** - علامة تحدد ما إذا كان يتم إعلام (**notified**) القارئ عندما تصبح قائمة الانتظار غير فارغة .

## Constructor Summary

`WaitFreeReadQueue(int maximum, boolean notify)`

ينشئ قائمة انتظار تحتوي على أقصى عدد من العناصر في الذاكرة الخالدة(immortal memory).

`WaitFreeReadQueue(int maximum, MemoryArea memory, boolean notify)`

تنشئ قائمة انتظار تحتوي على أقصى عدد من العناصر في الذاكرة.

`WaitFreeReadQueue(java.lang.Runnable writer, java.lang.Runnable reader, int maximum, MemoryArea memory)`

تنشئ قائمة انتظار تحتوي على أقصى عدد من العناصر في الذاكرة.

`WaitFreeReadQueue(java.lang.Runnable writer, java.lang.Runnable reader, int maximum, MemoryArea memory, boolean notify)`

تنشئ قائمة انتظار تحتوي على أقصى عدد من العناصر في الذاكرة.

# WaitFreeReadQueue

## Method Summary

| | |
|---|---|
| void | **clear**() <br> يقوم بتعيين الرتل إلى حالة فارغة. |
| boolea | **isEmpty**() <br> يستعلم عن قائمة الانتظار لتحديد ما إذا كان فارغًا. |
| n | **isFull**() <br> يستعلم النظام لتحديد ما إذا كان الرتل (قائمة الانتظار) ممتلئًا. |
| boolea | **read**() <br> يقرأ العنصر الذي تم إدراجه في الآونة الأخيرة من قائمة الانتظار ويعيده كنتيجة ، ما لم تكن قائمة الانتظار فارغة. |
| n | |
| java.lang.Object | |
| int | **size**() <br> يستعلم عن قائمة الانتظار لتحديد عدد العناصر فيها. |
| void | **waitForData**() <br> إذا كان الرتل كتلة فارغة انتظر حتى يقوم الكاتب writer بإدراج عنصر. |
| void | **write**(java.lang.Object object) <br> كتابة متزامنة وقابلة للحظر. |

public void **waitForData**() throws java.lang.InterruptedException
إذا كان الرتل كتلة فارغة انتظر حتى يقوم الكاتب writer بإدراج عنصر..

# *Example (WaitFreeReadQueue ):*

```
import javax.realtime.*;
public class Main extends RealtimeThread {
   public static WaitFreeReadQueue Box = new WaitFreeReadQueue(5, true);
   class FirstThread implements Runnable {
      public void run() {
         int count = 0;
         while ( count++ < 10 ) {
            System.out.println("FirstThread writing message");
            safeSend(count);
            try { Thread.sleep(10); } catch ( Exception e ) { }
         } safeSend(-1); }
      private void safeSend (final int count) {
         ImmortalMemory.instance().executeInArea(
         new Runnable() {
            public void run() {
               try {    if ( Box.isFull() )          System.out.println("FirstThread blocking");
                              if ( count == -1 )  Box.write("TERM");
                                else                    Box.write("Produced Msg " + count);
                        } catch ( Exception e ) { } } });
```

# *Example (WaitFreeReadQueue ):*

```
class SecondThread implements Runnable {
        public void run() {
            try { while ( true ) {
                    System.out.println(" SecondThread waiting");
                    Box.waitForData();
                    String item = (String)Box.read();
                    System.out.println(" SecondThread received: " + item);
                    if ( item.equalsIgnoreCase("TERM") ) return; } }
            catch ( Exception e ) { e.printStackTrace(); } } }  public
                void run() { startFirst();startsecond();  }
                private void startFirst() { new Thread( new FirstThread() ).start(); }
                private void startsecond() {
                    ImmortalMemory.instance().enter(
                     new Runnable() { public void run() {
                     int pri = PriorityScheduler.instance().getMaxPriority();
                    PriorityParameters sched = new PriorityParameters(pri);
                   new NoHeapRealtimeThread(sched, null,null, ImmortalMemory.instance(),null, new SecondThread() ).start(); } );  }
public static void main (String[] args) {
                                        new Main().start();  }}
```

TextPad - D:\AleppoUniversity Files\Dr.Mar...

File Edit Search View Tools Macros Configure Window Help

Output.txt

```
QProducer writing message
QProducer writing message
QProducer writing message
QProducer writing message
QProducer writing message
QProducer writing message
QProducer blocking
NHRT QConsumer waiting
  NHRT QConsumer received: Msg #1
NHRT QConsumer waiting
  NHRT QConsumer received: Msg #2
NHRT QConsumer waiting
  NHRT QConsumer received: Msg #3
NHRT QConsumer waiting
  NHRT QConsumer received: Msg #4
NHRT QConsumer waiting
  NHRT QConsumer received: Msg #5
NHRT QConsumer waiting
  NHRT QConsumer received: Msg #6
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #7
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #8
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #9
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #10
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #11
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #12
```

Dr. Marwa Dahdouh

Search Results | Tool Output

File Edit Search View Tools Macros Configure Window Help

Output.txt

```
  NHRT QConsumer received: Msg #31
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #32
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #33
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #34
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #35
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #36
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #37
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #38
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #39
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #40
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #41
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #42
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #43
```

Search Results

File Edit Search View Tools Macros Configure Window Help

Output.txt

```
QProducer writing message
  NHRT QConsumer received: Msg #89
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #90
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #91
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #92
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #93
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #94
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #95
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #96
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #97
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #98
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #99
NHRT QConsumer waiting
QProducer writing message
  NHRT QConsumer received: Msg #100
NHRT QConsumer waiting
  NHRT QConsumer received: TERM
```

Search Results

# *Example (WaitFreeWriteQueue ):*

```
import javax.realtime.*;
public class Main extends RealtimeThread {
public static WaitFreeWriteQueue queue = new WaitFreeWriteQueue(5);
   class QProducer implements Runnable {
      public void run() {
         int times = 0;
         while ( times++ < 100 ) {
            String s = "This is msg# " + times; queue.write(s);
            RealtimeThread.waitForNextPeriod();
         } queue.force("term"); }  }
   class QConsumer implements Runnable {
      public void run() { System.out.println("QConsumer waiting on the queue...");
         try { boolean loop = true;
            while ( loop == true ) {
               String msg = (String)queue.read();
             System.out.println("QConsumer received: " + msg);
               if ( msg.equalsIgnoreCase("term") )
                  loop = false; }  }
         catch ( Exception e ) { e.printStackTrace(); } } }
```

OutPut.txt

```
QConsumer waiting on the queue...
QConsumer received: This is msg# 1
QConsumer received: This is msg# 2
QConsumer received: This is msg# 3
QConsumer received: This is msg# 4
QConsumer received: This is msg# 5
QConsumer received: This is msg# 6
QConsumer received: This is msg# 7
QConsumer received: This is msg# 8
QConsumer received: This is msg# 9
QConsumer received: This is msg# 10
QConsumer received: This is msg# 11
QConsumer received: This is msg# 12
QConsumer received: This is msg# 13
QConsumer received: This is msg# 14
QConsumer received: This is msg# 15
QConsumer received: This is msg# 16
QConsumer received: This is msg# 17
QConsumer received: This is msg# 18
QConsumer received: This is msg# 19
QConsumer received: This is msg# 20
QConsumer received: This is msg# 21
QConsumer received: This is msg# 22
QConsumer received: This is msg# 23
QConsumer received: This is msg# 24
QConsumer received: This is msg# 25
QConsumer received: This is msg# 26
QConsumer received: This is msg# 27
QConsumer received: This is msg# 28
QConsumer received: This is msg# 29
QConsumer received: This is msg# 30
QConsumer received: This is msg# 31
QConsumer received: This is msg# 32
QConsumer received: This is msg# 33
QConsumer received: This is msg# 34
QConsumer received: This is msg# 35
QConsumer received: This is msg# 36
QConsumer received: This is msg# 37
```

```
public Main() { }
   public void run() {
                        startConsumer(); startProducer(); }
   private void startConsumer() {
                        new Thread( new QConsumer() ).start();
                        try { RealtimeThread.sleep(1); } catch ( Exception e ){ } }
   private void startProducer() { ImmortalMemory.instance().enter(
         new Runnable() { public void run() {
               PeriodicParameters rel = new PeriodicParameters( new RelativeTime(1,0));
               int pri = PriorityScheduler.instance().getMaxPriority();
               PriorityParameters sched = new PriorityParameters(pri);
               new NoHeapRealtimeThread(sched, rel,null, ImmortalMemory.instance(),null, new QProducer()
).start(); } }); }
   public static void main (String[] args) {
                        new Main().start();   }}
```