The Guarded Suspension Pattern

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand what condition variables are
- Note a human known use of condition variables
- Know what pattern condition variables implement

Executing the put method 
wakes up the waiting thread 
to continue the execution of 
the get method where it's 
suspended.

```
Message get_message () {
    // Acquire lock and try to get a message, if available.
    lock.acquire ();
    while (empty ()) // Suspend thread while queue is empty.
        not_empty.condition.wait ();
    Message m = get_message_impl (); // Get the message.
    ...
    lock.release (); // Release lock
}
```

```
Message put_message (Message m) {
    // Acquire lock and put a message into the queue.
    lock.acquire ();
    ...
    put_message_impl (m);
    // Wake up threads waiting to get a message.
    not_empty_condition.notify ();
    lock.release (); // Release lock.
}
Implementing Guarded Suspension with CVs
Implementing Guarded Suspension with CVs

- CVs are most often used to implement the *Guarded Suspension* pattern.

```
Message get_message () {
    # Acquire lock and try to get a message, if available.
    lock.acquire ();
    while (empty ()) # Suspend thread while queue is empty.
        not_empty.condition.wait ();
    Message m = get_message_impl (); # Get the message.
    # ...
    lock.release (); # Release lock
}
```

```
Message put_message (Message m) {
    # Acquire lock and put a message into the queue.
    lock.acquire ();
    # ...
    put_message_impl (m);
    # Wake up threads waiting to get a message.
    not_empty_condition.notify ();
    lock.release (); # Release lock.
}
```

**Require both a lock to be acquired & a precondition to be satisfied before an operation can be executed.**

See [en.wikipedia.org/wiki/Guarded_suspension](en.wikipedia.org/wiki/Guarded_suspension)
Implementing Guarded Suspension with CVs

- This pattern is applied to operations that can run only when a condition is satisfied.

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- This pattern is applied to operations that can run only when a condition is satisfied, e.g.,
  - a lock is acquired

A condition variable is always associated with a lock
This pattern is applied to operations that can run only when a condition is satisfied, e.g.,
- a lock is acquired
- a precondition holds

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

```
Lock l = new Lock()
Condition cond =
    l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

See www.youtube.com/watch?v=mJZZNHekEQw
Implementing Guarded Suspension with CVs

• In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied.

```
Lock l = new Lock();
Condition cond =
   l.newCondition();
...

l.lock()
while (conditionNotSatisfied())
   cond.await()
doOperationProcessing()
```

Note the tentative nature of “may”..
In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

First, a lock must be acquired.

```
Lock l = new Lock()
Condition cond =
    l.newCondition()
...
    l.lock()
    while (conditionNotSatisfied())
        cond.await()
    doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

Second, a condition is checked (in a loop) with the lock held.

```java
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
    doOperationProcessing()
```

ConditionVariable

- `await()`
- `signal()`
- `signalAll()`
In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

A condition can be arbitrarily complex.

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```
In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.

- A condition can be arbitrarily complex.

A method call, an expression that involves shared state, etc.

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

Any state shared between threads must be protected by a lock associated with the CV.
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex

<table>
<thead>
<tr>
<th>Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock()</td>
</tr>
<tr>
<td>unlock()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ConditionVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>await()</td>
</tr>
<tr>
<td>signal()</td>
</tr>
<tr>
<td>signalAll()</td>
</tr>
</tbody>
</table>

The calling thread will block (possibly repeatedly) while the condition is not satisfied (await() atomically releases the lock)

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```
Implementing Guarded Suspension with CVs

• In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied

• A condition can be arbitrarily complex

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```

Another thread can signal condition when shared state may now be true

`cond.signal()`
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied.
- A condition can be arbitrarily complex.

```
Lock l = new Lock();
Condition cond = l.newCondition();
...

l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperation Processing();
```

*`await()` reacquires the lock & condition is rechecked in loop*
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex
- Waiting on a CV releases the lock & suspends the thread *atomically*

```java
// Code example
Lock l = new Lock()
Condition cond = l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

The lock is released when the thread is suspended on the CV
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.
- A condition can be arbitrarily complex.
- Waiting on a CV releases the lock & suspends the thread \textit{atomically}.
- Thread $T_1$ is suspended until thread $T_n$ signals the CV.

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock();
while (conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied.
- A condition can be arbitrarily complex.
- Waiting on a CV releases the lock & suspends the thread *atomically*.
- Thread $T_1$ is suspended until thread $T_n$ signals the CV.

When a thread is signaled it wakes up & must re-acquire its associated lock.
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on may now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
  - Thread $T_1$ is suspended until thread $T_n$ signals the CV

```
Lock l = new Lock()
Condition cond =
    l.newCondition()
...

l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```

*After lock is re-acquired the thread can reevaluate its condition to see if it’s satisfied*
Implementing Guarded Suspension with CVs

- In this example thread T₁ uses a CV to suspend its execution until thread Tₙ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex
- Waiting on a CV releases the lock & suspends the thread *atomically*
- Thread T₁ is suspended until thread Tₙ signals the CV

If condition is not satisfied the thread must wait (which releases the lock atomically)

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
l.lock()
while (conditionNotSatisfied())
    cond.await()
doOperationProcessing()
```
Implementing Guarded Suspension with CVs

- In this example thread $T_1$ uses a CV to suspend its execution until thread $T_n$ notifies it that shared state it's waiting on *may* now be satisfied
- A condition can be arbitrarily complex
- Waiting on a CV releases the lock & suspends the thread *atomically*
- Thread $T_1$ is suspended until thread $T_n$ signals the CV

---

```java
Lock l = new Lock();
Condition cond = l.newCondition();
...
while (!conditionNotSatisfied())
    cond.await();
doOperationProcessing();
```

*After the lock is re-acquired & the condition is satisfied the operation can proceed (with lock held)*
End of the Guarded Suspension Pattern