15-410
“What could possibly go wrong?”

“Paradise Lost”
Feb. 7, 2020

Dave Eckhardt
Brian Railing
Outline

When to use `if()` vs. `while()`
Consider the lowly worker thread

/* note: not a thrgrp_*() worker thread */
void
worker(void *ignored)
{
    workitem *work;
    while (work = find_work())
        perform(work);
    thr_exit((void *) 0);
}
Consider the lowly worker thread

/* note: not a thrgrp_*() worker thread */
void worker(void * ignored)
{
    workitem *work;
    while (work = find_work())
        perform(work);
    thr_exit((void *) 0);
}

But a funny thing happens...
What's Wrong With This Picture?

```c
workitem *
find_work(void)
{
    workitem *w;
    mutex_lock(&m);
    if (going_out_of_business)
        w = (workitem *) 0;
    else
        w = (workitem *) dequeue(q);
    mutex_unlock(&m);
    return (w);
}
```
Better?

```c
mutex_lock(&m);
if (going_out_of_business) {
    w = (workitem *) 0;
} else {
    if (!(w = (workitem *) dequeue(q))) {
        cond_wait(&new_work, &m);
        w = (workitem *) dequeue(queue);
    }
}
mutex_unlock(&m);
return (w);
```
What We Hope For

<table>
<thead>
<tr>
<th>find_work()</th>
<th>queue_work()</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex_lock(&amp;m);</td>
<td>mutex_lock(&amp;m);</td>
</tr>
<tr>
<td>if (!...dequeue(..))</td>
<td>enqueue(..);</td>
</tr>
<tr>
<td>cond_wait(&amp;new, &amp;m);</td>
<td>cond_signal(&amp;new);</td>
</tr>
<tr>
<td>w = dequeue(..);</td>
<td>mutex_unlock(&amp;m);</td>
</tr>
<tr>
<td>mutex_unlock(&amp;m);</td>
<td></td>
</tr>
</tbody>
</table>
What Went Wrong?

What went wrong?
What Went Wrong?

What went wrong?
- Nothing!
What Went Wrong?

What went wrong?
  - Nothing!

But what if there is *an evil third thread*?
What About the “Evil Third Thread”?

mutex_lock(&m);
if (going_out_of_business) {
    w = (workitem *) 0;
} else {
    if (!(w = (workitem *) dequeue(q))) {
        cond_wait(&new_work, &m);
        w = (workitem *) dequeue(queue);
    }
}
mutex_unlock(&m);
return (w);
Not Exactly What We Hope For

<table>
<thead>
<tr>
<th>find_work()</th>
<th>queue_work()</th>
<th>find_work()</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(&amp;m);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if (!..deq(..))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cwait(&amp;new, &amp;m);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lock(&amp;m);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>enqueue(...);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>csignal(&amp;new);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unlock(&amp;m);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>lock(&amp;m);</td>
</tr>
<tr>
<td>w = deq(..);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unlock(&amp;m);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>return(w);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>return (0);</td>
</tr>
</tbody>
</table>
Have We Seen This Before?

What went wrong?

- Protected world state wasn't ready for us
- We blocked
- Somebody prepared the world for us to run
- We ran
  - We assumed nobody else had run
  - We assumed the world state was still ready for us

When have we seen this “happiness revocation”? 
To “if()” Or Not To “if()”?

mutex_lock(&m);
if (going_out_of_business) {
    w = (workitem *) 0;
} else {
    
    while (!(w = (workitem *) dequeue(q)))
        cond_wait(&new_work, &m);
}
mutex_unlock(&m);
return (w);
/* XXX still wrong! - rewrite after class */
Summary

if() vs. while()

- If somebody can revoke your happiness, you'd better check
Related Work

TOCTTOU

• ?
Related Work

“Toucan at Whipsnade Zoo”, William Warby, 2012-05-06, CC-BY
Related Work

**TOCTTOU**

- “Time of Check to Time of Use”
  - A standard “bug class”
  - Isn't that what we have here?
Related Work

**TOCTTOU**

- “Time of Check to Time of Use”
  - A standard “bug class”
  - Isn't that what we have here?
- “Correct, but wrong”
Related Work

**TOCTTOU**

- "Time of Check to Time of Use"
  - A standard “bug class”
  - Isn't that what we have here?
- “Correct, but wrong”
  - Many people think TOCCTOU bugs are always security bugs
  - Fundamentally, we expect the revoked condition to become unrevoked again (soon!)
  - Unlike the general case, this can be fixed in less than a line of code!