CSE 410: Systems Programming
Races and Synchronization

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo
Races

Races, or race conditions, are situations where:

- Two or more events are dependent upon each other
- Some of the events may happen in more than one order, or even simultaneously
- There exists some ordering of the events that is incorrect

For example:

- Some state will be updated multiple times
- Output will be produced based on the state

If some order of updates results in invalid output, this is a race.
Synchronization

Synchronization, in the context of a computer program, is the deliberate ordering of events via some mechanism.

There are many synchronization mechanisms, working in different ways.

Synchronization mechanisms may:

- Directly order events
- Simply ensure that events do not happen simultaneously
- Ensure that two events begin at the same time
- …

Synchronization is how we avoid races.
Race Conditions

The textbook defines a race as:

\[\text{ [...] when the correctness of a program depends on one thread reaching point } x \text{ in its control flow before another thread reaches point } y.\]

Note that there may be many points \(x\) and \(y\)!

The relationship between \(x\) and \(y\) may change over time, as well.

For example, “once thread \(T_1\) has reached point \(p\), it must reach point \(x\) before any other thread reaches point \(y\).”
Data Races

While data races, or races involving modification of data, are not the only kind of race, they are very common.

A data race occurs when:

- Two or more concurrent flows access shared state
- One or more of these flows modifies the state
- The order of the accesses/modifications is important
- The synchronization in use is insufficient to preserve the necessary order

Races among any number of concurrent flows for the same data may be reduced to a set of pairwise races.

At least one access in each pair must be a modifying operation.
Example Race

Consider two threads running the following code:

```c
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings++;
}
```

<table>
<thead>
<tr>
<th></th>
<th>T₁ index:</th>
<th>T₂ index:</th>
</tr>
</thead>
<tbody>
<tr>
<td>nstrings:</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>strings:</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Consider two threads running the following code:

```c
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings++;
}
```

**Thread T1:** Index: 0

**Thread T2:**
- nstrings: 0
- strings:
  - NULL
  - NULL
  - NULL
  - NULL
Example Race

Consider two threads running the following code:

```c
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings++;
}
```

<table>
<thead>
<tr>
<th>T1 index: 0</th>
<th>T2 index: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>nstrings: 0</td>
<td></td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>strings:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>
```
Example Race

Consider two threads running the following code:

```
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings++;
}
```

<table>
<thead>
<tr>
<th></th>
<th>T₁ index: 0</th>
<th>T₂ index: 0</th>
<th>nstrings: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>strings</td>
<td></td>
<td>T₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Example Race

Consider two threads running the following code:

```c
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings++;
}
```

<table>
<thead>
<tr>
<th>T1 index: 0</th>
<th>T2 index: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>nstrings: 0</td>
<td></td>
</tr>
<tr>
<td>strings:</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>
Example Race

Consider two threads running the following code:

```c
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings++;
}
```

T₁ index: 0
T₂ index: 0
nstrings: 1

strings:

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th>T₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2018 Ethan Blanton / CSE 410: Systems Programming
Example Race

Consider two threads running the following code:

```c
char *strings[4];
int nstrings;

void setstring(char *str) {
    int index = nstrings;
    strings[index] = str;
    nstrings ++;
}
```

```
T_1 index: 0
T_2 index: 0
nstrings: 2
strings:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_1</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

T_1 → ___________________________ ← T_2
Summary

...
Next Time …
References I

Required Readings

License

Copyright 2018 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see https://www.cse.buffalo.edu/~eblanton/.