15-213 Recitation 07

Race Conditions in Tshlab

Sean Stangl (sstangl@andrew.cmu.edu)

October 19, 2009

Outline

- Announcements / Questions
- Race Conditions
  - Abstract Discussion
  - Examples
- Race Conditions in Tshlab
  - waitpid()
  - Solution attempts

Public Service Announcements

- Tshlab due on Thursday (10/22).
  - With late days, on Saturday (10/24).
- Malloclab released Thursday night.
- Exam 2 is next Thursday (10/29).

Any questions (unrelated to tshlab)?

This recitation is very dense. The material is not simple.
Please ask questions as we go through it.

Imperative Language State

- Imperative language programmers make assumptions about execution ordering.
  - `foo(); bar(); baz();` has a known ordering.
- Each command modifies the *state* of the program.
  - Programmers also assume that the program is in a certain state *before* a command executes.
  - Each state assumption is called an **invariant**.
- This abstraction is a gigantic finite state machine.
- This abstraction is what is implicitly taught in 15-123.

An Extremely Abstract State Example

- Let `S(A)` mean "Global State A".
- Suppose we have the following functions:
  - `foo() : S(A)` $\longrightarrow$ `S(B)`
  - `bar() : S(B)` $\longrightarrow$ `S(C)`
  - `baz() : S(C)` $\longrightarrow$ `S(A)`
- What is the end state of this code?
  - `foo(); bar(); baz();`
- What is the end state of this code?
  - `foo(); foo(); bar(); baz();`
  - What invariants are violated?

What is a race condition, then?

- A race condition is when **concurrent** code asynchronously changes the program state in an unhandled, unexpected, or undesired manner.
  - We'll have a few examples, of course.
- Concurrent code is code executing in any one of:
  - Another thread (in the same process).
  - Another process (that can change the current process).
  - Signal handlers (ah-hah!).
  - ...
- On the previous slide, the second `foo();` could have come from another thread, or in a signal handler!

That was the best complete definition I could think of.

- But it's very abstract.
  - It may be useful to you when reviewing this recitation later.
- So we'll go through some simple examples of race conditions.
  - Think about how the program state has been unwantedly changed.

Consider the following threads, which modify a global `int sum=0`.

Adder Thread 1	Adder Thread 2
`add1():`	`add2():`
`int local = sum;`	`int local = sum;`
`local += 3;`	`local += 17;`
`sum = local;`	`sum = local;`

We run both these threads *at the same time*, and then print out the value of `sum`. What are possible values for `sum`?

What we want to have happen:

Time	Adder Thread 1	Adder Thread 2
0	`int local = sum; // 0`	
1	`local += 3; // 3`	
2	`sum = local; // 3`	
3		`int local = sum; // 3`
4		`local += 17; // 20`
5		`sum = local; // 20`

At the end, `sum == 20`.

What could go wrong (1):

Time	Adder Thread 1	Adder Thread 2
0	`int local = sum; // 0`	
1	`local += 3; // 3`	
2		`int local = sum; // 0`
3		`local += 17; // 17`
4		`sum = local; // 17`
5	`sum = local; // 3`	

At the end, `sum == 3`! Hm.

What could go wrong (2):

Time	Adder Thread 1	Adder Thread 2
0		`int local = sum; // 0`
1		`local += 17; // 17`
2	`int local = sum; // 0`	
3	`local += 3; // 3`	
4	`sum = local; // 3`	
5		`sum = local; // 17`

With this ordering, `sum == 17`.

How does this apply to Tshlab?

- We don't use threads in Tshlab.
  - But we do use signal handlers...
- Consider the code on the next slide, which deals with reaping finished child processes.
  - Imagine that we want to wait for the foreground process.
  - Why is the following approach wrong?

Consider this method of waiting for a foreground process.

eval(), Parent	SIGCHLD Handler

`if (fg) {`
`  waitpid(child, ...);`
`}`
	`pid = waitpid(.., WNOHANG ..);`
	`if (pid > 0) {`
	`  print_status(pid);`
	`}`

- Sometimes this works, and the child status is printed.
- Sometimes `pid == -1`, and no status is ever printed.
  - What invariant has been violated?
  - How can we make sure that the status is always printed?

Well, let's try without `waitpid()`.

eval(), Parent	SIGCHLD Handler

`if (fg) {`
`  while(joblist(child));`
`}`
	`pid = waitpid(.., WNOHANG ..);`
	`if (pid > 0) {`
	`  print_status(pid);`
	`  remove_from_joblist(pid);`
	`}`

- Does this fix the race condition?
- Is this a good idea?

Maybe this fixes the problem:

eval(), Parent	SIGCHLD Handler
`if (fg) {`	`pid = waitpid(.., WNOHANG ..);`
`while(joblist(child)) {`	`if (pid > 0) {`
`sleep(100);`	`print_status(pid);`
`}`	`remove_from_joblist(pid);`
`}`	`}`

- Does this work?
- Is this a good idea?

Questions?