Course 'Operating Systems Architecture' – tutorial: Thread APIs

UFAZ, L2
Lecturer: Pierre Parrend, Rabih Amhaz
This complementary material is made available by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau on:
http://pages.cs.wisc.edu/~remzi/OSTEP/
http://pages.cs.wisc.edu/~remzi/OSTEP/Homework/homework.html

In this section, we'll write some simple multi-threaded programs and use a specific tool, called **`helgrind`**, to find problems in these programs.
Read the `README` in the homework download for details on how to build the programs and run `helgrind`.

1. Exercise 1

First build `main-race.c`. Examine the code so you can see the (hopefully obvious) data race in the code. Now run `helgrind` (by typing `valgrind --tool=helgrind main-race`) to see how it reports the race.

1.1 Does it point to the right lines of code?

1.2 What other information does it give to you?

2. Exercise 2

2.1 What happens when you remove one of the offending lines of code?

2.2 Now add a lock around one of the updates to the shared variable, and then around both. What does `helgrind` report in each of these cases?

3. Exercise 3

Now let's look at `main-deadlock.c`. Examine the code. This code has a problem known as **deadlock** (which we discuss in much more depth in a forthcoming chapter).

3.1 Can you see what problem it might have?

4. Exercise 4

4.1 Now run `helgrind` on this code. What does `helgrind` report?

5. Exercise 5

Now run `helgrind` on `main-deadlock-global.c`. Examine the code.

5.1 Does it have the same problem that `main-deadlock.c` has?

5.2 Should `helgrind` be reporting the same error?

5.3 What does this tell you about tools like `helgrind`?

6. Exercise 6

Let's next look at `main-signal.c`. This code uses a variable (`done`) to signal that the child is done and that the parent can now continue.

6.1 Why is this code inefficient? (what does the parent end up spending its time doing, particularly if the child thread takes a long time to complete?)

7. Exercise 7

Now run `helgrind` on this program.

7.1 What does it report?

7.2 Is the code correct?

8. Exercise 8

Now look at a slightly modified version of the code, which is found in `main-signal-cv.c`. This version uses a condition variable to do the signalling (and associated lock).

8.1 Why is this code preferred to the previous version?

8.2 Is it correctness, or performance, or both?

9. Exercise 9

Once again run `helgrind` on `main-signal-cv`.

9.1 Does it report any errors?