cs154: Introduction to Computer Systems
Autumn 2019

Homework 7
(Assigned Nov 14)
Due Nov 18 (Monday) 11:59pm

Submit answers by committing a file into the `hw7` sub-directory of your `CNETID-cs154-aut-19` svn repository. Do not create this directory yourself: an "`svn update`" at the top level of your checkout will create it. The filename should be either `hw7.txt` or `hw7.pdf` for answers in plain ASCII text, or PDF, respectively. PDFs of scanned hand-written pages must not exceed 6 megabytes. No other file formats, or filenames are acceptable, and no files besides `hw7.txt` or `hw7.pdf` will be graded. Not following directions will result in losing points.

(1) (20 points)

(For this question, you should read the CSAPP book, Chapter 8.5.7 (or 8.5.6 in the third edition), "Synchronizing Flows to Avoid Nasty Concurrency Bugs").
Some race conditions plague the following code, which can be compiled and run on the CSIL Linux machines (with "`gcc -m32 -o race race.c; ./race`").

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <signal.h>
4
5  pid_t forkret = 0;
6
7  void handler1(int sig) {
8    printf("(%d) handler1\n", getpid());
9    if (!forkret) {
10     printf("(%d) seeya\n", getpid());
11     exit(0);
12   }
13 }
14
15 int main() {
16   printf("(parent) hello --------------\n");
17   signal(SIGUSR1, handler1);
18   if ((forkret = fork()) == 0) {
19     printf("(child) hello I'm %d\n", getpid());
20     kill(getppid(), SIGUSR1);
21     while (1) {};
22   } else {
23     printf("(parent) child is %d\n", forkret);
24     kill(forkret, SIGUSR1);
25     wait(NULL);
26     printf("(parent) child done\n");
27   }
28   exit(0);
29 }
```

A. Reading the code, and seeing that the child and the parent both send a `SIGUSR1` signal to each other after the `fork`, one might expect output like (knowing that PIDs differ between runs):

```
1 (parent) hello --------------
2 (parent) child is 101
3 (child) hello I'm 101
4 (100) handler1
5 (101) handler1
```

```
6  (101) seeya
7  (parent) child done
```

Give an account of each numbered line of output, by writing

- the number of the output line (no need to reproduce the output line itself), and then all in the same line:
- "P" or "C" for whether it came from the parent or the child process, respectively
- the line number from the C source code who's execution produced the output
- the value of `forkret`

B. One possible output of the program is:

```
1  (parent) hello --------------
2  (parent) child is 101
3  (child) hello I'm 101
4  (101) handler1
5  (101) seeya
6  (parent) child done
```

Give an account of each line of output like in part A, and then answer: what is the race condition that leads to this output instead of that in part A? That is, what are the steps in each of the child and parent processes that are in a race with each other, so that depending on which one finishes first, the output is either that from part A or part B? (We're using "step" here instead of "instruction" because in this case we can describe the race condition without getting down to the level of assembly, but we may need a finer level of granularity than lines of code.)

C. Another possible output of the program is:

```
1  (parent) hello --------------
2  (parent) child is 101
3  (101) handler1
4  (101) seeya
5  (parent) child done
```

Like part B: give an account of each output line, and describe the race condition and why it leads to this output.

D. Another possible output of the program is:

```
1  (parent) hello --------------
2  (child) hello I'm 101
3  (100) handler1
4  (100) seeya
```

Like part B: give an account of each output line, and describe the race condition and why it leads to this output.