CS 240 SI Worksheet #20
Valerie Zhao
4/24/17

Processes

1. Find the documentation for `waitpid(2)` (which you should read for the shell pset): https://linux.die.net/man/2/waitpid
   (can be googled or accessed through shell assignment page)

   a. What is the method signature?
   pid_t waitpid(pid_t pid, int *status, int options);

      i. How does the pid argument work?
      Manual says:

      The value of pid can be:

      < -1
      meaning wait for any child process whose process group ID is equal to the absolute value of pid.

      -1

      meaning wait for any child process.

      0

      meaning wait for any child process whose process group ID is equal to that of the calling process.

      > 0

      meaning wait for the child whose process ID is equal to the value of pid.

      ii. How are the options specified?
      Manual says:

      The value of options is an OR of zero or more of the following constants:

      WNOHANG
      return immediately if no child has exited.

      WUNTRACED

also return if a child has stopped (but not traced via ptrace(2)). Status for traced children which have stopped is provided even if this option is not specified.

… (there are more in the manual)

b. What does it do? Why is important?
It waits for the child process to terminate, reaps it (so that it doesn't use up more system resources), and then returns (and the parent process--the one that called waitpid--can continue execution).

c. When does it return?
When the child process terminates; if the child process has already terminated then it returns immediately.

d. What values does it return?
Straight from the manual: "on success, returns the process ID of the child whose state has changed; if WNOHANG was specified and one or more child(ren) specified by pid exist, but have not yet changed state, then 0 is returned. On error, -1 is returned."

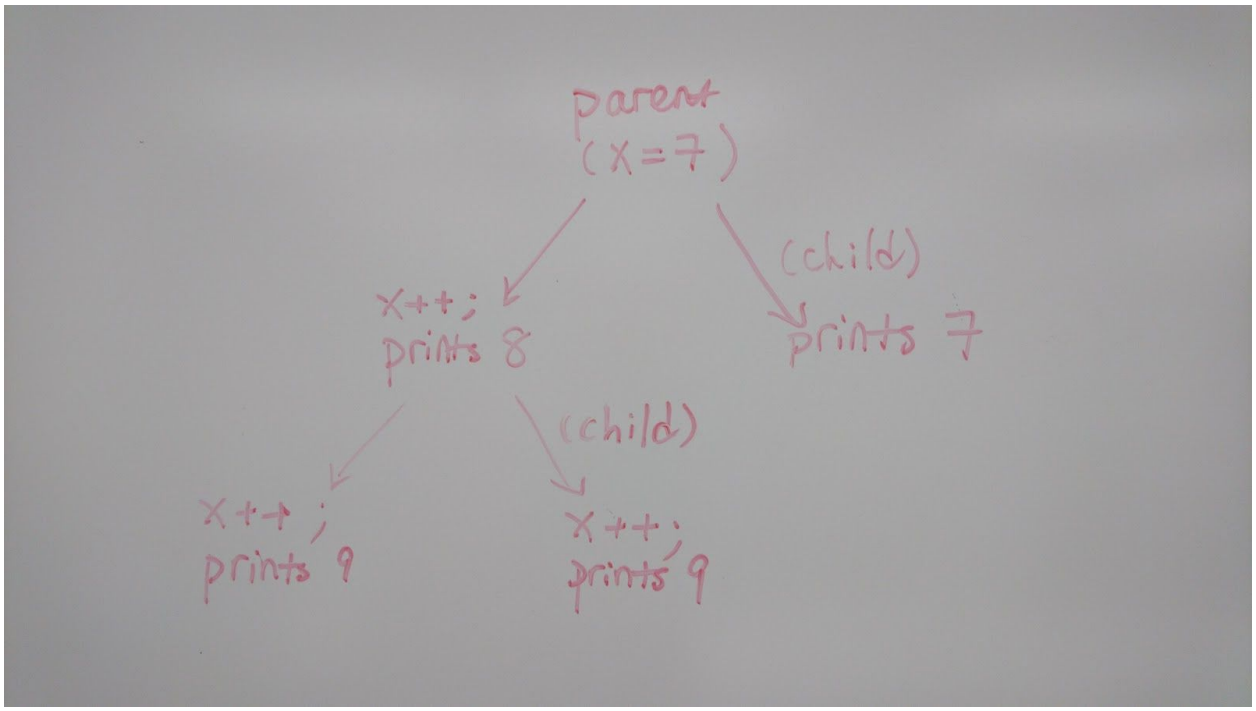
e. How is it useful for implementing a shell?
As stated on the assignment page, it should be used to check whether a process has terminated (with the WNOHANG option) *without waiting*, in order to check for and remove/reap any terminated background processes.

To carry out foreground jobs, other jobs must be paused; waitpid can be used to wait for the foreground job to finish.

*Adapted from CSE351 Winter 2016, Final Examination (March 16, 2016), Problem 5: https://courses.cs.washington.edu/courses/cse351/17sp/past-exams/cse351-16wi-final-solution.pdf

2. *List all possible outputs of the following code in C:

```
int x = 7;
if(fork()) {
      x++;
      printf(" %d ", x);
      fork();
      x++;
      printf(" %d ", x);
} else {
      printf(" %d ", x);
}
```



8 must be printed before the two 9s, but otherwise there is no restriction on the order of the printf()s, therefore there are 4 possible outputs:

7 8 9 9
8 7 9 9
8 9 7 9
8 9 9 7

*Adapted from CSE351 Winter 2016, Final Examination (March 16, 2016), Problem 5: https://courses.cs.washington.edu/courses/cse351/17sp/past-exams/cse351-16wi-final-solution.pdf