Lecture 05: Creating and Coordinating Processes, Take II

- Spawning and synchronizing with multiple child processes
  - A parent can call `fork` multiple times, provided it eventually reaps the child processes once they exit.
  - If we want to reap child processes as they exit without concern for the order they were spawned, then this does the trick (full program with error checking right here):

```c
int main(int argc, char *argv[]) {
    for (size_t i = 0; i < 8; i++) {
        if (fork() == 0) exit(110 + i);
    }
    while (true) {
        int status;
        pid_t pid = waitpid(-1, &status, 0);
        if (pid == -1) { assert(errno == ECHILD); break; }
        if (WIFEXITED(status)) {
            printf("Child %d exited: status %d\n", pid, WEXITSTATUS(status));
        } else {
            printf("Child %d exited abnormally.\n", pid);
        }
    }
    return 0;
}
```
Lecture 05: Creating and Coordinating Processes, Take II

- Spawning and synchronizing with multiple child processes
  - Note we feed a -1 as the first argument to `waitpid`. That -1 states we want to hear about any child as it exits, and pids are returned in the order their processes finish.
  - Eventually, all children exit and `waitpid` correctly returns -1 to signal there are no more processes under the parent's jurisdiction.
  - When `waitpid` returns -1, it sets a global variable called `errno` to the constant `ECHILD` to signal `waitpid` returned -1 because all child processes have terminated. Interestingly enough, that's the "error" we want.
Lecture 05: Creating and Coordinating Processes, Take II

- **Spawning and synchronizing with multiple child processes**
  - We can do the same thing we did in the first program, but monitor and reap the child processes in the order they are forked.
  - Check out the abbreviated program below (full program with error checking right here):

```c
int main(int argc, char *argv[]) {
    pid_t children[8];
    for (size_t i = 0; i < 8; i++) {
        if ((children[i] = fork()) == 0) exit(110 + i);
    }

    for (size_t i = 0; i < 8; i++) {
        int status;
        pid_t pid = waitpid(children[i], &status, 0);
        assert(pid == children[i]);
        assert(WIFEXITED(status) && (WEXITSTATUS(status) == (110 + i)));
        printf("Child with pid %d accounted for (return status of %d).\n", children[i], WEXITSTATUS(status));
    }

    return 0;
}
```
Lecture 05: Creating and Coordinating Processes, Take II

- **Spawning and synchronizing with multiple child processes**
  - This version spawns and reaps processes in some first-spawned-first-reaped manner.
  - Understand, of course, that the child processes aren't required to exit or otherwise terminate in FSFR order. In theory, the first child thread could finish last, and the reap loop could be held up on its very first iteration until the first child really is done. But the process zombies (as they're called) are certainly reaped in the order they were forked.
  - Below is a single sample run of the \texttt{reap-in-fork-order} executable. The content published within parentheses is the same with every test run. Only the pids change between runs, and even those are guaranteed to be published in increasing order.

```
myth60$ ./reap-as-they-exit
Child with pid 4689 accounted for (return status of 110).
Child with pid 4690 accounted for (return status of 111).
Child with pid 4691 accounted for (return status of 112).
Child with pid 4692 accounted for (return status of 113).
Child with pid 4693 accounted for (return status of 114).
Child with pid 4694 accounted for (return status of 115).
Child with pid 4695 accounted for (return status of 116).
Child with pid 4696 accounted for (return status of 117).
myth60$
```