Exceptional Control Flow

Annalise Tarhan

September 30, 2020

8 Homework Problems

8.9 For each of the given process pairs, indicate whether they run concurrently (Y) or not (N).

AB N

AC Y

AD Y

BC Y

BD Y

CD Y

8.10 Match each behavior with each function.

Called once, returns twice fork

Called once, never returns longjmp, execve (execve only returns if error)

Called once, returns one or more times setjmp

8.11 How many "hello" output lines does this program print?

Six. When i=0, there are two forks that print "hello." In each of those forks, i is incremented to 1, and each fork splits, printing one "hello" each.

8.12 How many “hello” output lines does this program print?

Eight. After the fork operations in doit, there are four processes. Each prints “hello” once in doit and again in main.

8.13 What is one possible output of the following program?

x=4 x=2 x=3

8.14 How many “hello” output lines does this program print?

Three. The original process is forked in the first line of doit and immediately returns, printing “hello” once in main before exiting. The child of that fork is also forked, and each of those two processes prints “hello” once in doit and exits without returning to main.

8.15 How many “hello” output lines does this program print?

Five. The original process is forked in the first line of doit and immediately returns, printing “hello” once in main before exiting. The child of that fork is also forked, and each of those two processes prints “hello” once in doit and again in main.

8.16 What is the output of the following program?

“counter = 2” The child process decrements its own copy of counter, not one that is visible to the parent process.

8.17 Enumerate all of the possible outputs of the program in Practice Problem 8.4.

Hello / 1 / 0 / Bye / 2 / Bye
Hello / 0 / 1 / Bye / 2 / Bye
Hello / 0 / Bye / 1 / 2 / Bye

8.18 Determine which of the following outputs are possible from the given program.

112002 Possible

211020 Not possible

102120 Possible

122001 Not possible

100212 Possible

8.19 How many lines of output does the following function print, in terms of n?

Assuming $n \geq 1$, the function prints "hello" 2^n times.

8.20 Use execve to write a program called myls, based on the /bin/ls program.

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[], const char* env[]) {
        if(execve("/bin/ls", argv, env) < 0) {
                printf("ERROR");
        }
        return 0;
}
```

8.21 What are the possible output sequences from the following program?

abc, bac

8.22 Write your own version of the Unix system function: int mysystem(char *command);

```
int mysystem(char *command) {
        if(fork() == 0) {
                char *args[]={"sh", "-c", command, NULL};
                if (execve("/bin/sh", args, NULL) < 0) {
                        printf("Command not found\n");
                }
                exit(-1);
        } else {
                int status;
                wait(&status);
                if (WIFEXITED(status)) {
                        return WEXITSTATUS(status);
                } else {
                        return status;
                }
        }
}
```

8.23 A child process sends a signal to its parent process each time an event occurs, telling the parent's signal handler to increment a global counter variable. When the parent calls printf, counter always has a value of 2, even though the child has sent five signals to the parent. Why?

The problem is that signals are not queued, so if a second signal is received soon enough after another of the same type, it will be ignored. Using signals this way is never safe, but this code is especially bad since the handler is slow.

8.24 Modify the program so that each child terminates abnormally after attempting to write to a location in the read-only segment and the parent prints output that is identical to the following example.

```
int main() {
        int status, i;
        pid_t pid;

        for (i = 0; i < N; i++) {
                if ((pid = Fork()) == 0) {
                        *s = 'a';
                        exit(100+i);
                }
        }

        while ((pid = wait(&status))) {
                if (pid == -1 && errno == EINTR) continue;
                if (pid == -1) break;

                if (WIFEXITED(status))
                printf(
                "child %d terminated normally with exit
                status=%d\n", pid, WEXITSTATUS(status)
                );

                else {
                char* stat_str = strsignal(status);
                int crawler = 0;
                while (stat_str[crawler]!=':') crawler++;
                stat_str[crawler] = '\0';
                printf(
                "child %d terminated by signal %d:
                %s\n", pid, status, stat_str);
                }
        }

        if (errno != ECHILD)
        unix_error("waitpid error");
        exit(0);
}
```

8.25 Write a version of the fgets function, called tfgets, that times out after 5 seconds. If the user doesn't type an input line within 5 seconds, tfgets returns NULL. Otherwise, it returns a pointer to the input line.

```
sigjmp_buf buf;

void handler(int sig) {
        siglongjmp(buf, 1);
}

char *tfgets(char *str, int n, FILE *stream) {
        if(!sigsetjmp(buf, 0)) {
                signal(SIGALRM, handler);
                alarm(5);
                return fgets(str, n, stream);
        } else {
                return NULL;
        }
}
```

8.26 Write a shell program that supports job control with the given features.

It's really hard to justify skipping this one. I have a couple excuses ready, but after spending an embarrassing number of hours working on it, I just don't want to anymore. I worked out about half of it on my own, then glanced at the solution. Turns out, I'd missed the entire concept of using sigsuspend to wait for signals. Should I go back and play with it until I at least understand it? Yes. Am I going to? No. As icky as it is to leave a project like this half-done and half-learned, I'm going to give myself a break this time. Thanks for listening.