HW-8-Signals

CS 503 Systems Basics

Problem 8.23

One of your colleagues is thinking of using signals to allow a parent process to count events that occur in a child process. The idea is to notify the parent each time an event occurs by sending it a signal and letting the parent's signal handler increment a global counter variable, which the parent can then inspect after the child has terminated. However, when he runs the test program in Figure 8.45 on his system, he discovers that when the parent calls printf, counter always has a value of 2, even though the child has sent five signals to the parent. Perplexed, he comes to you for help. Can you explain the bug?

The bug is due to the both the concurrent nature of the Fork call and the delay in the signal handler. Within the parent, a signal handler is first set up. After the fork is made, the child process sends 5 signals to the parent while the parent waits for the child to finish. As the child sends the signal, the parent process's waiting gets interrupted and it jumps immediately to the signal handler, which increments the counter by 1 and sleeps. However, during the 1 second delay, the child process is still running and continues to send more signals to the parent before the handler has a chance to finish and handle the next one. As a result, those signals are not counted and instead are just lost by the parent.

Problem 8.24

Modify the program in Figure 8.18 so that the following two conditions are met:
1. Each child terminates abnormally after attempting to write to a location in the read-only text segment.
2. The parent prints output that is identical (except for the PIDs) to the following:

```
child 12255 terminated by signal 11: Segmentation fault
child 12254 terminated by signal 11: Segmentation fault
```

Hint: Read the man page for psignal(3).

```
#include "csapp.h"
#define N 2
int main()
 {
 int status, i;
 pid_t pid;
 /* Parent creates N children */
 for (i = 0; i < N; i++)
    if ((pid = Fork()) == 0){ /* child */
        char *text = "read-only text";
        text[2] = 't'; /* Try write to read-only text segment (segfault) */
        exit(100+i);
    }
```

```
/* Parent reaps N children in no particular order */
while ((pid = waitpid(-1, &status, 0)) > 0){

   if (WIFEXITED(status))
       printf("child %d terminated normally with exit status=%d\n", pid,
              WEXITSTATUS(status));
   else{
       char str[200]; /* Store error string*/
       sprintf(str, "child %d terminated by signal %d", pid,
               WTERMSIG(status));
       psignal(WTERMSIG(status), str); /* Print signal string */
   }
}
/* The normal termination is if there are no more children */
if (errno != ECHILD)
   unix_error("waitpid error");
exit(0);
}
```

Problem 8.24 Additional Question

Modify your solution to Problem 8.24 so that one (and only one) child installs a Segmentation-fault handler which prints an error message and exits. What is the output of the program after this change?

```
#include "csapp.h"
#define N 2

void segFaultHandler(int sig){
     perror("Caught Segmentation Fault");
     exit(0);
}


int main()
 {
 int status, i;
 pid_t pid;
 /* Parent creates N children */

 for (i = 0; i < N; i++)

   if ((pid = Fork()) == 0){ /* child */
       if (i == 0) /* Let the first child create the handler */
           Signal(SIGSEGV, segFaultHandler);

       char *text = "read-only text";
       text[2] = 't'; /* This will cause segmentation fault */
       exit(100+i);
   }
 /* Parent reaps N children in no particular order */

 while ((pid = waitpid(-1, &status, 0)) > 0){
```

```
    if (WIFEXITED(status))
        printf("child %d terminated normally with exit status=%d\n",
pid,WEXITSTATUS(status));
    else{
        char str[200]; /* Store error string*/
        sprintf(str, "child %d terminated by signal %d",
pid,WTERMSIG(status));
        psignal(WTERMSIG(status), str); /* Print signal string */
    }
 }
 /* The normal termination is if there are no more children */
 if (errno != ECHILD)
    unix_error("waitpid error");
 exit(0);
 }
```

Output:

Caught Segmentation Fault: Success
child 155 terminated normally with exit status=0
child 156 terminated by signal 11: Segmentation fault

The output would look like this. The first child process still terminates normally because the signal handler caught the segmentation fault error. However, the second child will terminate abnormally because there is no handler to catch the error.