Homework H5: Chapter 8 Signals

CS 283 Systems Programming

Grading: 100 points

The following exercises come from the main text, Computer Systems: A Programmer's Perspective, 3rd Edition (CSAPP3). (33 points each)

- CSAPP3 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?

- CSAPP3 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).

- CSAPP3 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?

What to hand in:

Please submit the solutions to the problems above by the submission method specified in the Syllabus. Make sure your answers are concise and clear. Make sure you completely explain all your answers. We will deduct points if it is not obvious how you got your answer -- even if your answer is correct.

```
/* Figure 8.18 */
#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 */
    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
       printf("child %d terminated abnormally\n", pid);
   }

  /* The normal termination is if there are no more children */
  if (errno != ECHILD)
   unix_error("waitpid error");

  exit(0);
 }
```

```
/* Figure 8.45 */
#include "csapp.h"

int counter = 0;

void handler(int sig)
 {
  counter++;
  sleep(1);
  /* Do some work in the handler */
  return;
 }

int main()
 {
  int i;

  Signal(SIGUSR2, handler);

  if (Fork() == 0)
   {
    /* Child */
    for (i = 0; i < 5; i++)
     {
      Kill(getppid(), SIGUSR2);
      printf("sent SIGUSR2 to parent\n");
     }
    exit(0);
   }

  Wait(NULL);
  printf("counter=%d\n", counter);
  exit(0);
 }
```