ECE 434: Intro to Computer Systems -- Spring 2019 Instructor: Maria Striki

Homework 2 (45 points, 10+8+15+12)

Issue Date: Tue March 19th Due Date: Sat March 30th 2019

Note: Handled by groups of 4 OR 5 students, submit one document per group (via sakai or hardcopy).

Your Names and Contribution per student:

Problem 1: (10 pts)

The Sieve of Eratosthenes is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the multiples of 2.

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

The sieve of Eratosthenes can be expressed in pseudocode, as follows:

Input: an integer $n > 1$.

```
Let A be an array of Boolean values, indexed by integers 2 to n,

initially all set to true.

for i = 2, 3, 4, ..., not exceeding √n:
  if A[i] is true:
    for j = i², i²+i, i²+2i, i²+3i, ..., not exceeding n:
      A[j] := false.

Output: all i such that A[i] is true.
```

This algorithm produces all primes <= n. It includes common optimization, which is to start enumerating the multiples of each prime i from i^2. The time complexity of this algorithm is $O(n \log \log n)$.

Write a multi-threaded program that outputs prime numbers. The program should work as follows: the user will run the program and enter a number on the command line. The program creates a thread that outputs all the prime numbers less than or equal to the number entered by the user. Also, create a separate thread that outputs this subset of the above primes that have the following property: the number that is derived by reversing the digits is also prime (e.g., 79 and 97).

Solution:

Problem 2: (8 pts)

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ...
Formally, it can be expressed as:
$fib0 = 0$
$fib1 = 1$
$fibn = fibn-1 + fibn-2$

Write a multithreaded program that generates the Fibonacci sequence. This program should work as follows: On the command line, the user will enter the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, the parent thread will have to wait for the child thread to finish. This problem requires the parent thread to wait for the child thread to finish its execution before printing out the computed values. If we let the parent thread access the Fibonacci numbers as soon as they have been computed by the child thread—rather than waiting for the child thread to terminate—what changes would be necessary to the solution for this exercise? Implement your modified solution.

Solution:

Problem 3: (15 pts)

Consider that the system calls do not fail, function *f* () never returns, *f* () does not execute system calls. System calls can be interrupted by incoming signals. Since *f* () never returns, the processes generated by the program come to a permanent state: each process is under a specific function or system call.

Part 1: (5 pts) Answer the following (related) questions:

1. (1 pt) In the sys call `n = read (fd, bf, arg)` what does argument arg contain? What values may variable *n* have after its return from `read();`

2. (1 pt) What does the sys call wp = wait (& status) do? Suppose that after process with PID 412 has made that system call, then the following apply: wp == 618, WIFSIGNALED (status) == 1, WTERMSIG (status) == 3. Assuming that process 412 determines process 618, what happens to process 618 and with what call has process 412 affected it?

3. (1 pt) What happens to a process that calls `read ()` in an empty pipe when the write end remains open (for writing)?

4. (2 pts) What happens to a process that calls read after the last writer has exited and what happens to a process that calls write after the last reader has exited?

Part 2: (10 pts) Draw the process tree in its final form, i.e., when all processes have reached a steady state. Explain briefly how it arises.

At each node of the tree process please highlight: (i) the system call or a function where the PC of the corresponding process is located, (ii)) the program line from which the call took place.

The PIDs assigned by the system to the new processes increment by 1. You can assume that the initial process has PID = 1000.

Complete the process tree to demonstrate IPC: for each data transfer or signaling, draw a dotted arrow from the sender to recipient process. Above the arrow report the value that is transferred each time.

```
1  int pg[2]; int status;
2
3  void handler(int signum)
4  { int arg; arg = getpid(); kill(arg+2, 9); wait(&status); exit(5); f(0, -1); }
5
6  int main(void)
7  {
8       int i, j, n;
9       char c[4];
10      pid_t pid[4];
11
12      signal(SIGUSR1, handler);
13
14      pipe(pg);
15      for (i = 0; i < 4; i++) {
16              pid[i] = fork();
17              if (pid[i] == 0) {
18                       if (i == 1) sleep(5);
19                      n = read(pg[0], c, 2);
20                      for (j = 0; j < i; j++) {
21                      pid[i] = fork();
22                      if (pid[i] == 0)
23                      f(getppid(), j);
24                    }
25              for (j = 0; j < i; j++)
26              wait(NULL);
27              exit(2);
28             }
29      }
30
31      kill(pid[1], SIGUSR1);
32      for (i = 0; i < 4; i++) {
33              wait(&n);
34              write(pg[1], c, WEXITSTATUS(n));
35      }
36      f(i, -1);
37      return 0;
38 }
```

Solution:

Problem 4: (12 points)

Part 1 (6 pts): Write a program in C-like PSEUDOCODE that generates an output **as close as possible** to the following desired operation:
From main function we spawn 3 Pthreads. We are interested in handling signals both from main function and from the threads that belong to the following list:

`SIGINT, SIGSTOP, SIGILL.`

Each Pthread is spawned with the objective to catch one of the above signals only, and block or ignore the rest. Moreover, each thread with identity `tid[i]` should also compute the sum of integers from 0 to 1,000 x `tid[i]`. When one Pthread (or the main function) catches a signal, it should modify handling to just output the signal number and its own identity (thread or main).
The main function is set up to ignore the signals until the three threads are created. After the three threads join, then the main function restores default operation for all three signals.
In your pseudo code you may use either function signal or sigaction for setting up the signal handlers. Assume that signal does not automatically re-installs itself.

Part 2 (6 pts):
After you write the pseudo code, now please answer the following questions:
Q1: (3 pts) What will be observed when the process receives one of the above signals externally (from terminal) during execution? Briefly describe each case. What does the result depend on? Please justify your answer.
Q2: (3 pts) What will be observed when one thread sends one of the signals to the rest of the two threads and to the process itself? Study each case separately. Please justify your answer.

Solution: