

Program 1 Test
CSI4337

Compile and test on wind.ecs.baylor.edu

Compile with: `gcc -g -Wall -ansi -pedantic -std=c99`

-2 points per warning up to -5 for basic warnings

Run with: `valgrind --tool=memcheck --show-reachable=yes --leak-check=full --track-fds=yes ./<executable>`

-2 points for leaked file descriptors up to -5 (ignore fd=0, 1, and 2)

-2 points for each memory leak up to -5

Part I:

Program (30 points)

0 points if program fails to compile

If program fails to operate properly, deduct 50% (min) and grade by inspection

Verify error output goes to stderr

Test with no/2 parameters (-5 points for incorrect behavior or no usage message)

Test with 3 processes

Discussion (0 points)

For the discussion problems, I'm expecting something like the following. A (correct) claim with no justification is worth 2 points.

1. Processes are being run on multiple processors concurrently. When the program is run with two child processes, it takes the same amount of time to complete as when it's run with just one child process. This is because each child is running concurrently on its own CPU. The program only starts taking more time when it uses more child processes than CPUs. Wind has 4 CPUs; therefore, with five or more child processes, it is not able to run all children concurrently, so they must take turns using the available processors and the execution time grows accordingly. Here's my run

```
% time ./a.out 1
3.416u 0.001s 0:03.41 100.0%    0+0k 0+0io 0pf+0w
% time ./a.out 2
6.840u 0.003s 0:03.42 200.0%    0+0k 0+0io 0pf+0w
% time ./a.out 3
10.255u 0.001s 0:03.44 297.9%    0+0k 0+0io 0pf+0w
% time ./a.out 4
13.663u 0.003s 0:03.43 398.2%    0+0k 0+0io 0pf+0w
% time ./a.out 5
17.077u 0.003s 0:05.14 332.1%    0+0k 0+0io 0pf+0w
```

2. Each process may run on any CPU. If we have N CPUs and a particular process always ran on the same CPU, the execution time for N+1 children would be twice the single process execution time (or more). For N=4 with 5 children, two (or more) processes would run on one CPU while the other CPUs would have only one of the child processes. With two processes sharing the same CPU, they would run at about half the speed, and the program would take twice as long to complete (compared to a single child process). However, we see that the 5-child case takes about 1.5 times as long as the fewer-than-5-child case. The 5 children must be taking turns on the four available CPUs.

Part II (70 points):

0 points if program fails to compile

If program fails to operate properly, deduct 50% (min) and grade by inspection

Verify error output goes to stderr

Test 10! with 3 processes. Verify completely correct answer

Test 40! with 1 and 40 processes. Verify correct answer

Test N=-1 and P=1 – Bad N

Test N=0 and P=1 – Answer = 1

Test N=5 and P=7 – Bad P

If submission includes f.c, replace with prototype and penalize 10 points.

Compile using the following f(x) in a file named powf.c

```
double f(int x) {  
    return 5.3;  
}
```

Use N=5 and P=2. This should compute 5.3^{N+1}

Compile using the following f(x) in a file named doublef.c

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

```
double f(int x) {  
    static unsigned int executions = 0;  
    if (executions++ > 0) {  
        fprintf(stderr, "Warning: f(x) executed %u times\n", executions);  
    }  
    return 0;  
}
```

This should not print a warning for N=10 and P=2.

For both parts:

Take off points for poor coding practices.

Verify correct use of system calls such as fork() and wait() including test of return values

MARK unused include

- 10 for poor commenting
- 2 for each unjustified numeric constant
- 5 bad submission
- 1 for each section of commented code
- 5 if violate coding convention