CAL POLY
SAN LUIS OBISPO

CSC 357: Systems Programming

FINAL EXAM ASSIGNMENT

- Max points: 100.
- Use your laptop/workstation.
- I affirm that I neither will give nor receive unauthorized assistance on this examination.

Final Task:

Submit: finals.zip

The final submission should contain all your source code plus the executables with the corresponding names outlined below!

Happy coding and a wonderful (though) short break to all of you!

Task 1: Timed Execution of a Program (20 points)

Write a program **`countdown`** that starts any other program of your choice after a certain time has passed. While waiting for the "launch" of the chosen program, output a countdown that shows one period for every 0.5 seconds that are remaining before the launch. Hence, if the time to launch is 3 seconds, output

```
.....  //3 seconds until START
..... //2.5 seconds until START
.... //2 seconds until START
... //1.5 seconds until START
.. //1 second until START
. //0.5 seconds until START
// START of other program
```

How to:

You **must** use the system call **`setitimer`** to generate the SIGALRMs for each clock tick (Read the man pages!).
Because the display requires half-second accuracy, the granularity of the timer must be finer than one tick per second.
Remember that **`stdio`** is buffered. If you write something to the console with putchar() or printf(), it may not appear immediately. You should either use unbuffered writes, or flush the output stream after writing.

I will not test unexpected input; hence you may assume the command-line is well formed.

Example how I will run your code:

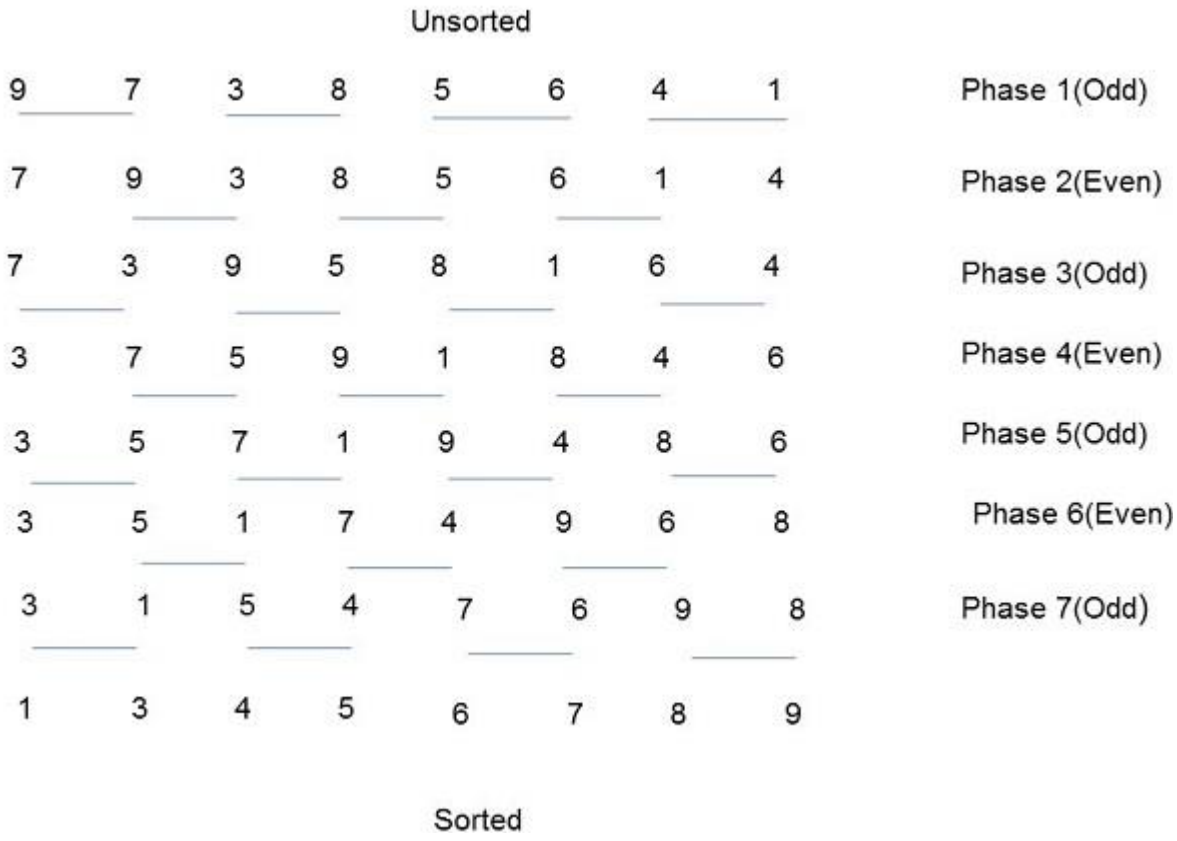
```
./countdown 3 ./myotherprogram

./countdown 4.5 ./programwithcommandlineargs hi 34
```

Task 2: Parallel Sorting Algorithm (80 points)

Write an Odd-Even Sort using different processes (not programs- you **must** use **fork()**).

This sort compares two adjacent numbers and switches them, if the first number is greater than the second number to get an ascending order list. The opposite case applies for a descending order series. Odd-Even transposition sort operates in two phases − **odd phase** and **even phase**. In both the phases, processes exchange numbers with their adjacent number in the right.
(See: https://www.tutorialspoint.com/parallel_algorithm/parallel_algorithm_sorting.htm)



Assume that the array is consisting of integers and passed via stdin one number at a time.

To make life easier, use stdin- redirection and prepare different files for testing different arrays.
You program should print the original array and the sorted array together with the time it took to sort it.

Example how I will run your code:

```
./EvenOdd 4 < test1.txt
```

Assuming the `text1.txt` file contains:

9 7 3 8 5 6 4 1

Possible output (feel free to format differently):

```
Initial Array: [ 9, 7, 3, 8, 5, 6, 4, 1]
Sorted Array:  [ 1, 3, 4, 5, 6, 7, 8, 9]

Processes: 4
Time to Sort: 23 ms
```

In the example above, your program will use 4 individual processes to sort the array, i.e. each process is in that case only comparing one value pair at each stage of the sort.

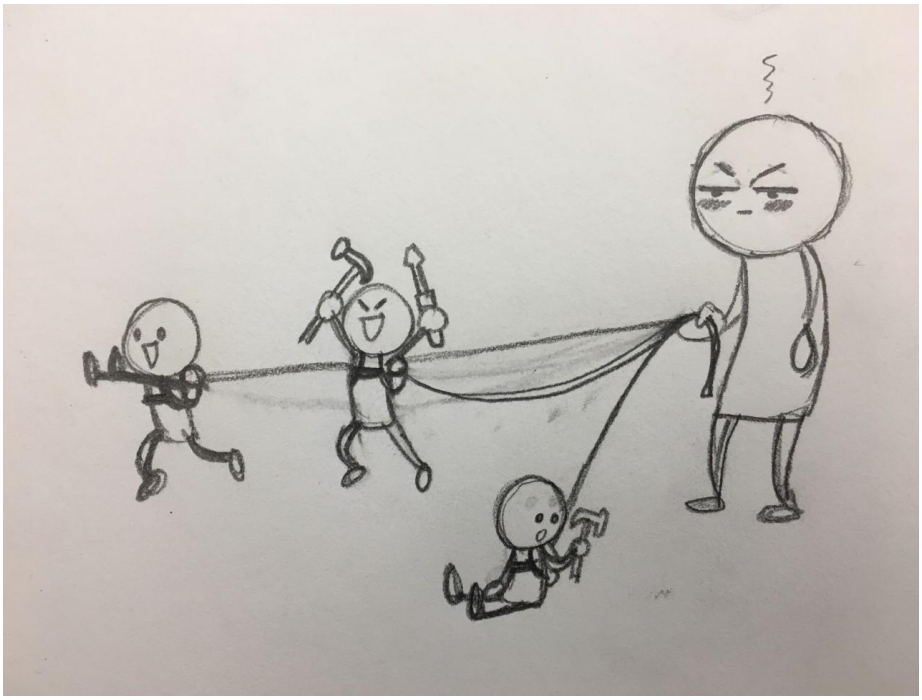
If the number of processes of an array of size 8 is larger than 4, you can cut at 4 processes.

If the number of processes for the example above is 3, divide as you may see fit.

If the number of processes for the example above is 2, every process will compare 2 value pairs at each stage.

Art by Gabriella Santiago, graduate from the CIA minor (Computing for Interactive Arts) and LAES at Cal Poly. She painted that for CPE 357 Eckhardt's sections but also wanted to share with you.

"Parent process and working child processes."



"Lock/Unlock"