CSE130 Winter 2020 : Assignment 1

In this assignment you will implement a multi process merge sort using the `fork()` and `wait()` system calls.

This lab is worth 5% of your final grade.

Submissions are due NO LATER than 23:59, Sunday January 26 2020 (1+ weeks)

Setup

SSH in to one of the two CSE130 teaching servers using your CruzID Blue password:

```
$ ssh <cruzid>@noggin.soe.ucsc.edu   ( use Putty http://www.putty.org/ if on Windows )
Or  $ ssh <cruzid>@nogbad.soe.ucsc.edu
```

Authenticate with Kerberos: ***(do this every time you log in)***

```
$ kinit      ( you will be prompted for your Blue CruzID password )
```

Authenticate with AFS: ***(do this every time you log in)***

```
$ aklog
```

Create a suitable place to work: ***(only do this the first time you log in)***

```
$ mkdir -p CSE130/Assignment1
$ cd CSE130/Assignment1
```

Install the lab environment: ***(only do this once)***

```
$ tar xvf /var/classes/CSE130/Winter20/Assignment1.tar.gz
```

Build the starter code:

```
$ cd ~/CSE130/Assignment1      ( always work in this directory )
$ make
```

Run the provided single process merge sort:

```
$ ./sort -s 32
```

Run the skeleton multi process merge sort:

```
$ ./sort -m 32     ( this will fail to sort the randomly generated array )
```

Also try:

```
$ make check       ( runs the required functional and non-functional tests - see below )
$ make grade       ( tells you what grade you will get - see below )
```


Additional Information

We covered process creation and control using the `fork()` and `wait()` system calls in class. Consult the lecture handouts to remind yourself how they work.

Note that whilst one operating system provided facility you will need to complete this assignment has been mentioned in class, it will not be covered in detail until Friday January 17. You can, however, make a lot of progress on this assignment without that additional piece of information, so I suggest you start as soon as possible.

Requirements

Basic:

- You have implemented a multi process merge sort that correctly sorts random arrays of integers when using the supplied `merge()` function

Advanced:

- Your implementation is at least 1.5 times faster than the supplied single process merge sort when using the supplied `merge()` function

What steps should I take to tackle this?

Come to sections and ask.

How much code will I need to write?

A model solution that satisfies all requirements adds approximately 35 lines of executable code.

Grading scheme

The following aspects will be assessed:

1. (100%) **Does it work?**

    a. Functional tests pass (45%)
    b. Non-Functional (performance) tests pass (45%)
    c. Your implementation is free of compiler warnings (10%)

2. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

    b. Your submission is determined to be a copy of a past or present student's submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 33% copied code No deduction
        - 33% to 66% copied code (-50%)
        - > 66% copied code (-100%)

What to submit

In a command prompt:

```
$ cd ~/CSE130/Assignment1
$ make submit
```

This creates a gzipped tar archive named `CSE130-Assignment1.tar.gz` in your home directory.

****** UPLOAD THIS FILE TO THE APPROPRIATE CANVAS ASSIGNMENT ******

§

