CSE130 Spring 2021 : Assignment 3

In this assignment you will use concurrency primitives to synchronize activity between POSIX threads.

This lab is worth 5% of your final grade.

Submissions are due NO LATER than 23:59, Monday May 3 2021 (1 week)

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
or    $ ssh <cruzid>@olaf.soe.ucsc.edu
or    $ ssh <cruzid>@thor.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/Assignment3
      $ cd CSE130/Assignment3
```

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

```
      $ tar xvf /var/classes/CSE130/Assignment3.tar.gz
```

Build the starter code:

```
      $ cd ~/CSE130/Assignment3      ( always work in this directory )
      $ make
```

Then try:

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

Run the incomplete manpage executable:

```
      $ ./manpage
```

Run the incomplete Cartman executable:

```
      $ ./cartman -s     ( simple configuration, will not generate a race condition )
      $ ./cartman -d     ( deadlock configuration, will generate a race condition )
      $ ./cartman -r     ( random configuration, may generate a race condition )
```


Additional Information

In Assignment 2 you merge sorted an array in multiple POSIX threads. In this assignment you will create and synchronize the activity of multiple threads to complete tasks requiring in-sequence and non-interleaved execution. You will need to use POSIX concurrency primitives to achieve this.

`lock`	https://linux.die.net/man/3/pthread_mutex_lock
`semaphore`	http://man7.org/linux/man-pages/man7/sem_overview.7.html
`condition variable`	https://linux.die.net/man/3/pthread_cond_signal etc.

Requirements

Basic / Manpage:

- Synchronize the activity of multiple threads to print a simplified manpage for Dykstra's Semaphores.

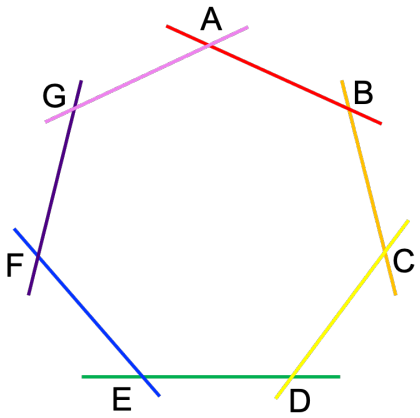
  Paragraphs for a simple manpage are made available to multiple threads is pseudo random order, these threads must synchronize their activity so they invoke a supplied method to display paragraphs in the correct order.

  Incorrect or no synchronization will see paragraphs displayed out of sequence and/or interleaved with each other. Both will cause the test to fail.

Advanced / Cartman:

- Protect access to critical sections with POSIX concurrency primitives.

  In an automated factory, Continuous Automated Rolling Trolleys (CARTs) running on narrow gauge tracks are used to move finished goods from the assembly areas to shipping. The products are fragile, so instead of CART tracks passing over and under each other, they cross at a small number of junctions as shown below.



  Your task is to write the CART Manager (Cartman) that allows CARTs to safely cross the small sections of track between junctions. These sections are small and CARTs move slowly, so in order for a CART to safely pass between two junctions, exclusive access to both junctions must be secured before allowing a CART to enter the critical section of track.

  Whenever a CART arrives at a junction, Cartman will be informed of the arrival via the `arrive()` callback function and must ensure safe transit across the critical section of track by first securing exclusive access to the relevant junctions via the `reserve()` function. Once exclusive access is gained, Cartman should call the `cross()` function to set he CART on its way. Once across the critical section of track, the junctions should be marked as available again by calling `release()`.

What steps should I take to tackle this?

Basic / Manpage:

- To get all seven paragraphs, each must be requested by a separate thread by calling the supplied functiom `getParagraphId()` which returns the paragraph each thread has been allocated. Once all paragraphs are obtained, the threads must agree between themselves who will call `showParagraph()` first, then second, then third, and so on.

Advanced / Cartman:

- Use concurrency primitives to ensure exclusive access to the critical sections of track. Watch out for deadlocks, they can occur at any point if you are careless, but running `cartman -d` is guaranteed to generate a race condition so check you can create a deadlock then implement an avoidance strategy.

  Note that whilst the `reserve()` and `release()` functions are 'thread safe' insofar as multiple threads can safely call them without internal data corruption, they do NOT provide any form of concurrency protection, your code must do that using some or all of the concurrency primitives we have focused on in class.

How much code will I need to write?

A model solution that satisfies all requirements has approximately 100 lines of executable code.

Grading scheme

The following aspects will be assessed:

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

   a. Manpage (30%)
   b. Simple Cartman (20%)
   c. Deadlock Cartman (20%)
   d. Random Cartman (20%)
   e. Your implementations are free of compiler warnings and memory errors (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:

      - < 25% copied code No deduction
      - 25% to 50% copied code (-50%)
      - > 50% copied code (-100%)

What to submit

In a command prompt:

```
$ cd ~/CSE130/Assignment3
$ make submit
```

This creates a gzipped tar archive named `CSE130-Assignment3.tar.gz` in your home directory.

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

