# Extra Credit Assignment - Concurrency and Synchronization

CS 536 - Operating Systems
Indiana University

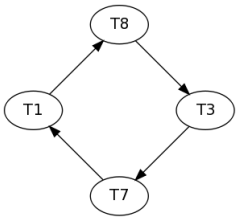
Spring 2016

## 1 Introduction

This assignment will cover the following topics:

- Concurrency primitives (mutexes and condition variables)
- The *pthreads* library
- Resource management in a multithreaded environment
- Implementing a message passing system on top of a shared memory model

The code given to you will, after several missing pieces are filled in, create some number of threads. Each of these threads will receive a pointer to their own *mailbox* struct that can be used to *send* and *receive* messages, as well as a pointer to another thread's *mailbox*. As well, they each have a pseudorandomly assigned *id* that ranges between 0 and $5 * NUM\_THREADS$. At first, these threads will be arranged in a ring. Your goal is to transform this ring into a sorted list of threads.




(a) How the threads are arranged originally.


(b) How the threads should end up.

## 2 Setup

This assignment can be completed on any POSIX compatible operating system. The main software requirements are *pthreads* and *gcc* (you may use another compiler if you wish).

To view the graphs that are generated by *tring* you will need to install the Graphviz package. To install this package on a Linux machine you can issue one of the following commands:

```
# Debian/Ubuntu command
sudo apt-get install graphviz

# Fedora command
sudo dnf install graphviz
```

## 3 The Assignment

Generally speaking, the goal of this assignment is as stated above in the Introduction. Specifically, when the main thread of execution sends the `PRINT` and `PING` messages to the threads they must process them in a linear order. The global variable *first_mb* will need to point to the lowest numbered thread's *mailbox*, and each thread after that should be arranged in increasing order of *id*, with the last thread's *next_mb* variable set to the value *NULL*. As well, your code should not leak memory. To test this last condition use the following command:

```
valgrind ./tring N
```

You may make changes to the following files on the condition that your changes to not break any of the provided code.

- mailbox.c
- Makefile
- tring.c
- tring_thread.c
- tring_thread.h

In several of these files you will find specific sections, labeled with `FIXME`, that you will need to provide code for. A short description of the code that needs to go there is provided in the comments.

One of these sections encompasses the entire body of the function *tring_start*. You may feel free to put most anything in this function, but it must handle several messages in a particular fashion. Specifically,

some short while after it is created each thread will receive two messages, `ID` and `MAILBOX`, in that order. The first contains the *id* of that thread, while the second contains the *mailbox* of the next thread in the ring.

Your code must also handle the `PING`, `PRINT`, and `SHUTDOWN` messages correctly. In response to a `PING` message a thread must call the *pong* function, passing its *id* as the argument. When a `PRINT` message is received your thread must, in some way, cause a line to be printed to the *tring.dot* file (using the *tring_print* function) that describes the edge between it and the thread pointed too by *next_mb*. Upon the arrival of a `SHUTDOWN` message a thread must free all of its resources and exit.

At several points the main thread of execution pauses, *pthread_cond_wait()*ing, giving the threads time to finish certain tasks. The first of these pauses is after the call to *tring_protocol_start*, at which point it is waiting for the threads to re-arrange themselves. It pauses again after it sends a `PING` and a `PRINT` message to the thread pointed to by *first_mb*. To wake the main thread of execution at each of these points, use the function *tring_signal*.

One of your first steps in completing this assignment should be making the functions *mailbox_send* and *mailbox_receive* threadsafe, as they are currently not.

# 4 Grading

This assignment will be graded as follows following bases: 20% for compiling, 20% for no deadlocks, 20% for proper thread cleanup, 20% for not leaking any memory, and 20% for correct output. We will be running your code with up to 500 threads, so please test your code thoroughly.

Any attempt to fool the code responsible for reporting success, through calling `pong` or `tring_print` in the wrong order, or other methods, will be considered cheating and result in a zero for the assignment.