Semaphores

Defining a Semaphore

- Invented by Edsgar Dijkstra
- A Semaphore is an integer with the following properties
  - When you create a semaphore you can initialise it to any integer value but after that you can only perform two operations on it. You can increment by one or decrament it by one.
  - You cannot read the current value of a semaphore
  - When a thread decrements a semaphore, if the result is negative, the thread blocks itself and cannot continue until another thread increments the semaphore
    - And the semaphore value is no longer negative
  - When a thread increments a semaphore, if there are other threads waiting on that semaphore then one of them becomes unblocked

Consequences of Definition

- When you signal a semaphore you do not necessarily know whether another thread is waiting, so the number of unblocked threads may be zero or one
- In general there is no way of knowing whether a thread will block on a decrement operation
- After an increment operation both the incrementing thread and one waiting thread can run concurrently – but there is no way of knowing which (if either) will continue immediately

Consequences

- The value of a semaphore indicates:
  - Positive integer represents the number of threads that can decrement without blocking
  - Negative integer represents the number of waiting (blocked) threads
  - Zero means no threads are waiting
- But you are not allowed to ask a semaphore what its value is!

Why use Semaphores

- They impose constraints that help programmers avoid errors
- Code using semaphores tends to be clean and organised
- Semaphores have efficient implementations
- Mainly we use them to force you to think clearly about the issues of concurrency
  - Lessons learned here will be applicable to any concurrency programming model you use in the future

Creating Semaphores in C++

- Semaphore.h, Semaphore.cpp are online
  - We will view them now!

Signaling with Semaphores

- A single semaphore can be used to send a signal from one thread to another
  - To indicate something has happened
- Use a semaphore initialised with value 0
- Thread waiting for signal calls wait
- Thread sending signal calls signal

Simple Rendezvous

- Generalised Signal Pattern
  - Thread 1 has to wait for thread 2
  - Thread 2 has to wait for thread 1
    - Both have to arrive at a certain point before aither proceeds
  - Thread A
    - A1;
    - A2;
  - Thread B
    - B1;
    - B2;
  - A1 must finish before B2 starts
  - B1 must finish before A2 starts

Exercise

- Implement both solutions in C++
- Create Makefile
- Document with Doxygen
- Put colution up on github