Completely Fair Scheduling (CFS)

The completely fair scheduler used by Linux is an implementation of the Weighted Round Robin Scheduling algorithm. The basic rundown of the algorithm is

1. Set the time it should take to go through all the threads, adjustable by the sys admin.
2. Given a weight for each thread, allocate time to the thrad in its proportion to it's weight against the sum of all weights.
3. There is a lower bound on how little time a thread can get allocated. If a threads runtime risks getting below this, increase the time we have to go through all the threads.

As an example, suppose we have a round time of 6 ms, with thread 1 that has weight 1024 and thread 2 has weight 335, thread 1 gets roughly $(1024/(1024 + 335)) * 6ms \approx 4.5ms$ runtime, while thread 2 has roughly runtime $(335/(1024 + 335)) * 6ms \approx 1.5ms$

Niceness and weight

Linux has the concept of nicenesss of a thread. It is an integer and the smaller the umber, the nicer the thread is. The niceness gives rise to the threads weight by (1024/(1.5^{niceness})). So in the above example, thread 1 has niceness 0, the other niceness 5. Note that if they had niceness 10 and 15, their ratio of their weights is the same.

Virtual runtime

Calculating how much a threads total runtime is compared to the sum of all runtimes can be a challenge. However, if we know that the runtime r_2 of thread 2 is a third of the runtime r_1 of thread 1, we get $(1/3)r_1 = r_2 \iff r_1 = 3r_2$

Hence, we say in this case that the for each unit of real runtime on thread 2, it has 3 units of virtual runtime.

In general, if a thread has weight w, and the sum of all weights are W, then a threads virtual runtime = W/w real runtime.

Combining it all

All threads are organised in a red-black tree with regard to their virtual runtime. The thread with lowest virtual runtime is the one chosen to run, and will get a time slice propotional to its weight in regard to the total sum of weights. Then

we calculate its virtual runtime, add it to the total virtual runtime and put it back in the tree, unless it is done executing.

A small detail on sleeping threads. Periodically, we will add some virtual runtime to a sleeping thread, such that they are never to far behind and so that they wont consume all resources once they wake up.