Thread Scheduler Efficiency Improvements for Multicore Systems

Daniel Collin Frazier

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

18 November 2017
UMM, Minnesota

Introduction

- The problem of thread scheduling has been around since the 1960s and for a while, scheduling implementation was largely unchanged
- The thread scheduler is an important system component that is always running
- The thread scheduler must be as efficient as possible
  - Reduce time spent doing necessary/maintenance tasks (the critical section)
- Increasing hardware requirements in the early 2000s made the problem more complex

“And you have to realize that there are not very many things that have aged as well as the scheduler. Which is just another proof that scheduling is easy.”
Linus Torvalds, 2001 [2]

Overview

- Establish needed concepts
- Thread scheduling and thread load-balancing on Linux
- Bug fixes and two new developments to the Linux thread scheduler

Before we define what thread scheduling is,

let’s get some terms out of the way!

Outline

- Establish needed concepts
  - Threads, Multithreading Programs
  - Parallel programming
  - Synchronicity and Locks
- Thread scheduling and thread load-balancing on Linux
- Bug fixes and two new developments to the Linux thread scheduler

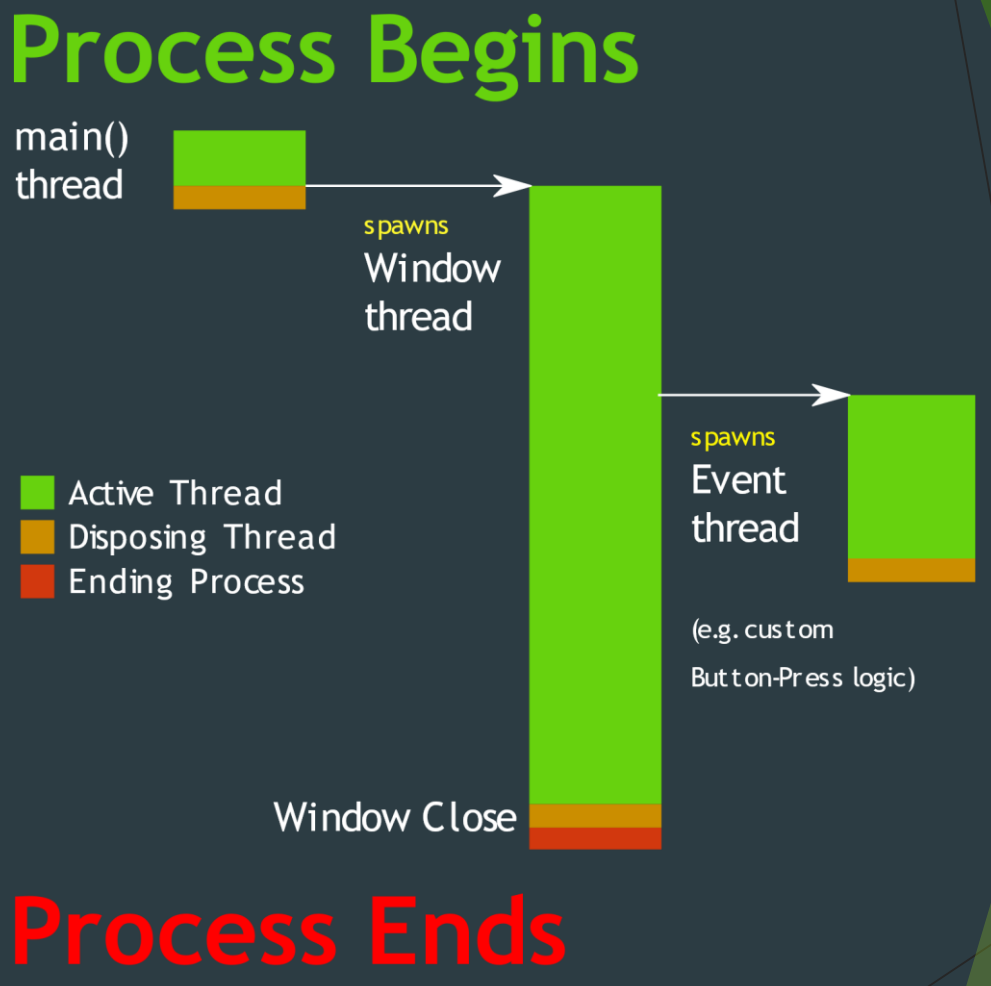
Using Threads

- Threads allow a program to run more than one independent task at one time

- Useful for…
  - Programs with long, mostly-independent computations
  - Programs with graphical interface

- Example GUI Program (right)

  In this figure there are three Threads created within one Process



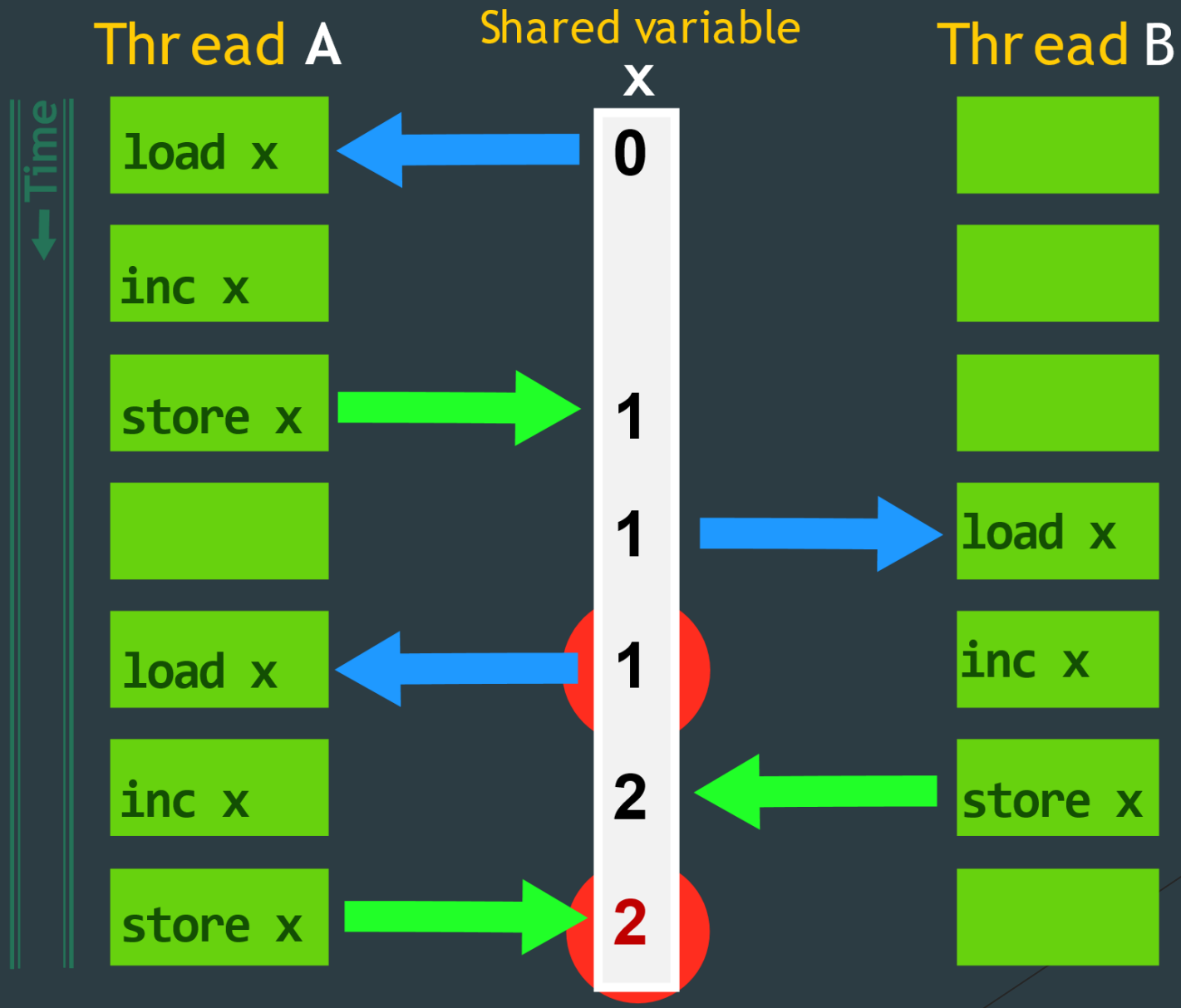
Using Threads

- A multithreaded program is a program that employs threads
- Concurrent computing techniques are techniques that allow many tasks to occur at the same time [W]
- Parallel computing techniques are techniques that allow many calculations to occur at the same time [W]
- Problems can be solved or improved using none, either or both of these techniques at once

Parallel Programming

- [Fill in once I actually know anything about parallel computing]
- Don’t go in to detail, high level descriptions.

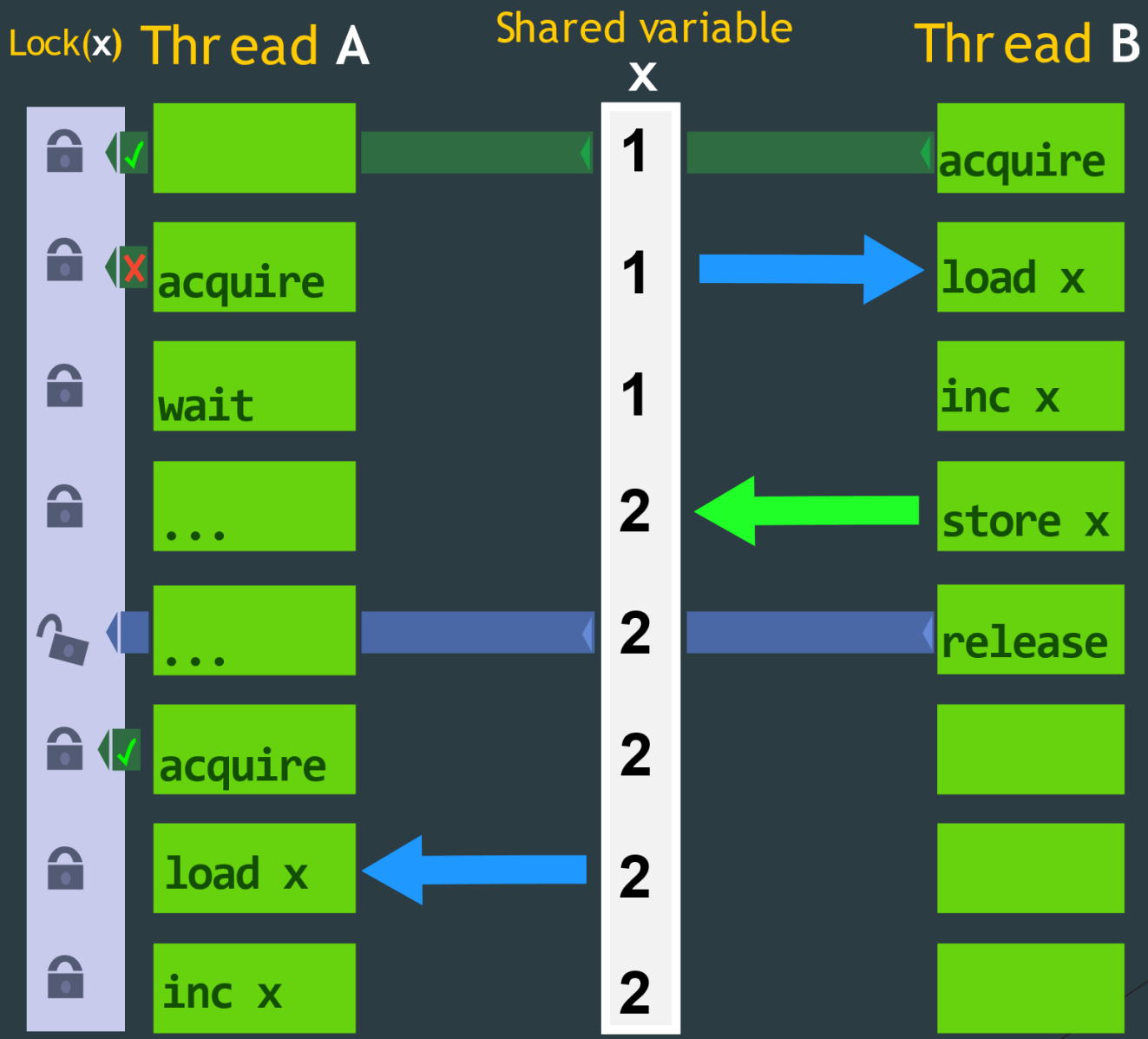
Race Condition (example)



Synchronicity and Locks

- Synchronicity can achieved by employing locks
- When locks are used properly, they fix race conditions
  - (Does this count for all abstract “systems”? Does it fix all types of race conditions? The abstract “lock” is engineered to solve race conditions for an abstract “system” right?)
- Locks are effectively booleans that are set when using objects or data shared between threads
- When a thread sets a lock to *locked*, that thread “acquires” the lock
- When a thread unlocks a lock, that thread “releases” the lock

Lock Example



Outline

- Establish needed concepts
- Thread scheduling and load-balancing on Linux
  - Completely Fair Scheduler (CFS)
  - Cache and Scheduling Domains
  - Load Balance algorithm for the CFS
- Bug fixes and two new developments to the Linux thread scheduler

Completely Fair Scheduler (CFS)

- Default Linux Thread Scheduler (there are others)
- Handles which threads are executed at what times and on which CPU cores
- Spend a *fair* amount of runtime on all threads

- The CFS implementation is simple for a single-core system
- For now, let's assume one single core CPU

- The scheduler switches active threads by saving and restoring thread and processor state information.
- Switching active threads and processes are called context switches.

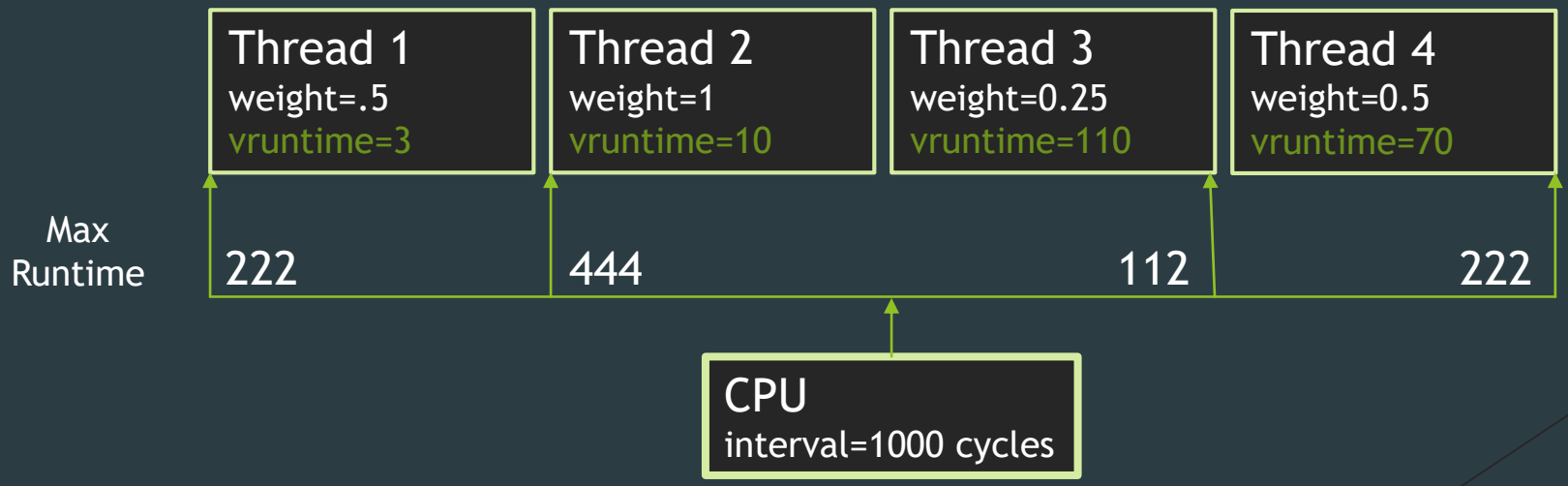
Process and Thread state

- Process
  - [state info]
- Thread
  - [state info]

Completely Fair Scheduler (CFS)

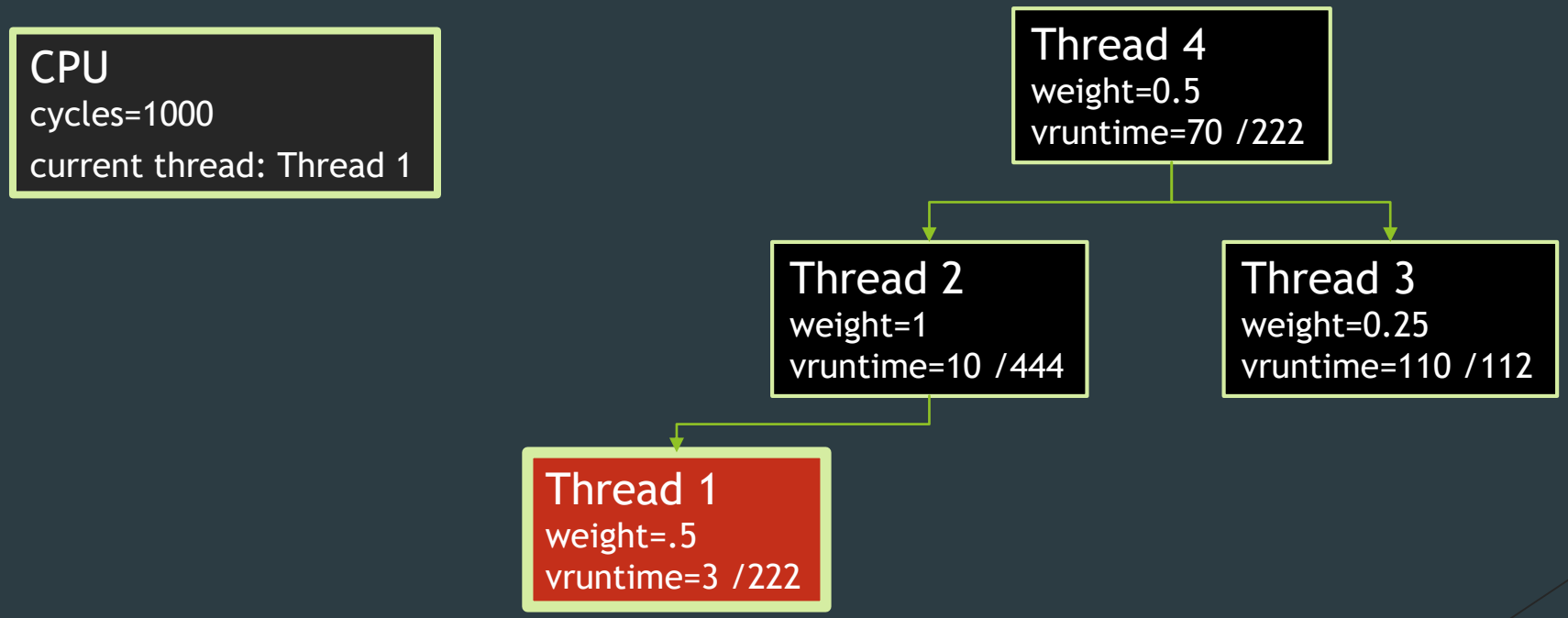
- Implementation of Weighted Fair Queueing (WFQ) scheduling algorithm

Goal: * Run all threads at least once within an arbitrary interval of CPU cycles
* "Timeslice" cycles evenly, prioritizing higher weights



Runqueue on One CPU

- When a thread is running, it accumulates *vruntime*
- A runqueue is a self organizing *Red-Black Tree* that sorts on vruntime
- New threads are chosen as the leftmost node, with the least vruntime.



Multiple Runqueues

- Context Switching is expensive across cores
- Multicore systems need multiple runqueues to minimize context switches

Outline

- Establish needed concepts
- Thread scheduling and thread load-balancing on Linux
- Bug fixes and two new developments to the Linux thread scheduler
  - Four bugs found within current implementation of CFS
  - FLSCHED: The lockless thread scheduler
  - Shuffler: Cache locality improvements via thread migration

Lock Contention

Conclusions

References

1. K. Kumar, P. Rajiv, G. Laxmi, and N. Bhuyan.
   Shuffling: A framework for lock contention aware thread scheduling for multicore multiprocessor systems.
   In 2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT), pages 289–300, Aug 2014.
2. J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, and A. Fedorova.
   The linux scheduler: A decade of wasted cores.
   In Proceedings of the Eleventh European Conference on Computer Systems, EuroSys ’16, pages 1:1–1:16, New York, NY, USA, 2016. ACM.
3. U. B. Nisar, M. Aleem, M. A. Iqbal, and N. S. Vo.
   Jumbler: A lock-contention aware thread scheduler for multi-core parallel machines.
   In 2017 International Conference on Recent Advances in Signal Processing, Telecommunications Computing (SigTelCom), pages 77–81, Jan 2017.