

CS 370: OPERATING SYSTEMS

[CPU SCHEDULING]

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Can you force an OS to fail by flooding it with a several small jobs and a few large jobs?
- Batch vs Interactive systems: what are they?
- Does the CPU ever make scheduling decisions?
 - ▣ No! Scheduling decisions are made entirely by the kernel
- How does the CPU know how long a process will run?



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.2

2

Topics covered in this lecture

- Scheduling Algorithms
 - ▣ SJF
 - ▣ Priority Scheduling
 - ▣ Round robin scheduling
- Multilevel feedback queues
- Lottery scheduling



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.3

3



SJF IN SCHEDULERS

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

4

Use of SJF in long term schedulers

- Length of the process time limit
 - ▣ Used as CPU burst estimate
- Motivate users to accurately estimate time limit
 - ▣ Lower value will give faster response times
 - ▣ Too low a value?
 - Time limit exceeded error
 - Requires resubmission!



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.5

5

The SJF algorithm and short term schedulers

- **No way to know** the length of the next CPU burst
- So, try to **predict** it
- Processes scheduled **based on predicted** CPU bursts



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.6

6

Prediction of CPU bursts: Make estimates based on past behavior

- t_n : Length of the n^{th} CPU burst
- τ_n : Estimate for the n^{th} CPU burst
- α : Controls weight of recent and past history
- $\tau_{n+1} = \alpha t_n + (1-\alpha) \tau_n$ Exponentially weighted moving average (EWMA)
- Burst is predicted as an **exponential average** of the measured lengths of previous CPU bursts



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.7

7

α controls the relative weight of recent and past history

- $\tau_{n+1} = \alpha t_n + (1-\alpha) \tau_n$
- Value of t_n contains our most recent information, while **τ_n stores the past history**
- $\tau_{n+1} = \alpha t_n + (1-\alpha) \alpha t_{n-1} + \dots + (1-\alpha)^j \alpha t_{n-j} + \dots + (1-\alpha)^{n+1} \alpha \tau_0$
- α is less than 1, $(1-\alpha)$ is also less than one
 - ▣ **Each successive term has less weight than its predecessor**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.8

8

The choice of α in our predictive equation

- If $\alpha = 1/2$
 - ▣ Recent history and past history are **equally weighted**
- With $\alpha = 1/2$; successive estimates of τ

$$t_0/2 \quad t_0/4 + t_1/2 \quad t_0/8 + t_1/4 + t_2/2 \quad t_0/16 + t_1/8 + t_2/4 + t_3/2$$
 - ▣ By the 3rd estimate, weight of what was observed at t_0 has dropped to $1/8$.



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
 COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.9

9

An example: Predicting the length of the next CPU burst

CPU burst (t_i)		6	4	6	4	13	13	13
"Guess" (τ_i)	10	8	6	6	5	9	11	12



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
 COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.10

10

The choice of α in our predictive equation

- $\tau_{n+1} = \alpha t_n + (1-\alpha) \tau_n$
- If $\alpha \rightarrow 0$, $\tau_{n+1} = \tau_n$
 - ▣ Current conditions are transient
- If $\alpha=1$, $\tau_{n+1} = t_n$
 - ▣ Only most recent bursts matter
 - ▣ History is assumed to be old and irrelevant



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.11

11

Time management is an oxymoron. Time is beyond our control, and the clock keeps ticking regardless of how we lead our lives. Priority management is the answer to maximizing the time we have.

John C. Maxwell

A photograph of a map with several red pushpins stuck into it. The pushpins are of different sizes and are scattered across the map. The map shows streets and some text, but it is out of focus.

PRIORITY SCHEDULING

12

Priority Scheduling

- **Priority** associated with each process
- CPU allocated to process with **highest** priority
- Can be preemptive or nonpreemptive
 - If preemptive: Preempt CPU from a lower priority process when a higher one is ready



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

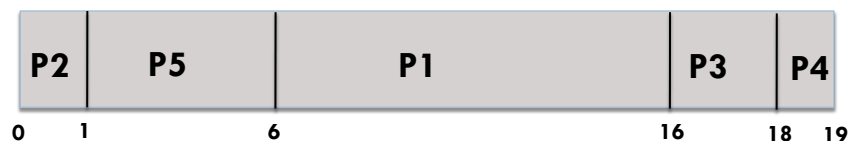
L15.13

13

Depiction of priority scheduling in action

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Here: Lower number means
higher priority



$$\text{Wait time} = (6 + 0 + 16 + 18 + 1) / 5 = 8.2$$



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.14

14

How priorities are set

- Internally defined priorities based on:
 - ▣ **Measured** quantities
 - ▣ Time limits, memory requirements, # of open files, ratio (averages) of I/O to CPU burst
- External priorities
 - ▣ Criteria outside the purview of the OS
 - ▣ Importance of process, \$ paid for usage, politics, etc.



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.15

15

Issue with priority scheduling

- Can leave lower priority processes waiting indefinitely
- Perhaps apocryphal tale:
 - ▣ MIT's IBM 7094 shutdown (1973) found processes from 1967!



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.16

16

Coping with issues in priority scheduling: Aging

- **Gradually increase priority** of processes that wait for a long time
- Example:
 - ▣ Process starts with a priority of 127 and decrements every 15 minutes
 - ▣ Process priority becomes 0 in no more than 32 hours



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.17

17

Can SJF be thought of as a priority algorithm?

- Priority is **inverse** of CPU burst
- The larger the burst, the lower the priority
 - ▣ Note: The number we assign to represent priority levels may vary from system to system



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.18

18



19

Round-Robin Scheduling

- Similar to FCFS scheduling
 - ▣ **Preemption** to enable switch between processes
- Ready queue is implemented as **FIFO**
 - ▣ Process Entry: PCB at *tail* of queue
 - ▣ Process chosen: From *head* of the queue
- CPU scheduler goes around ready queue
 - ▣ Allocates CPU to each process *one after the other*
 - CPU-bound up to a maximum of 1 **quantum**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.20

20

A round-robin analogy

- Hyperkinetic student studying for multiple exams simultaneously
 - ▣ If you switch between paragraphs of different textbooks? [Quantum is too short]
 - You won't get much done
 - ▣ If you never switch? [Quantum is too long]
 - You never get around to studying for some of the courses



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.21

21

Round Robin: Choosing the quantum

- Context switch is **time consuming**
 - ▣ Saving and loading registers and memory maps
 - ▣ Updating tables
 - ▣ Flushing and reloading memory cache
- What if quantum is 4 ms and context switch overhead is 1 ms?
 - ▣ 20% of CPU time thrown away in administrative overhead



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.22

22

Round Robin: Improving efficiency by increasing quantum

- Let's say quantum is 100 ms and context-switch is 1 ms
 - ▣ Now wasted time is only 1%
- But what if 50 concurrent requests come in?
 - ▣ Each with widely varying CPU requirements
 - ▣ 1st one starts immediately, 2nd one 100 ms later, ...
 - ▣ The last one may have to wait for 5 seconds!
 - ▣ A shorter quantum would have given them better service



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.23

23

If quantum is set longer than mean CPU burst?

- **Preemption will not happen very often**
- Most processes will perform a blocking operation before quantum runs out
- Switches happens only when process blocks and cannot continue



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.24

24

Quantum: Summarizing the possibilities

- Too short?
 - ▣ Too *many* context switches
 - ▣ *Lowers* CPU efficiency
- Too long?
 - ▣ *Poor* responses to interactive requests



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.25

25

Every inch of sky's got a star
Every inch of skin's got a scar
I guess that you've got everything now
Every inch of space in your head
Is filled up with the things that you read
I guess you've got everything now
And every film that you've ever seen
Fills the spaces up in your dreams
That reminds me
...
Every song that I've ever heard
Is playing at the same time, it's absurd
And it reminds me, we've got everything now
Everything Now, Arcade Fire



MULTI-LEVEL FEEDBACK QUEUES (MFQ)

Most commercial OS including Windows and MacOS, use this scheduling algorithm

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

26

MFQ is designed to achieve several simultaneous goals

- **Responsiveness:** Run short tasks quickly as in SJF
- **Low Overhead:** Minimize number of preemptions, as in FIFO
 - ▣ Minimize time spent making scheduling decisions
- **Starvation-Freedom**
 - ▣ All tasks should make progress, as in Round Robin
- **Background tasks**
 - ▣ Defer system maintenance tasks, such as defragmentation, so they do not interfere with user work
- **Fairness**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.27

27

Does MFQ achieve all of these?

- As with any real system that must **balance** several, conflicting goals ...
 - ▣ MFQ does not perfectly achieve these goals simultaneously
- MFQ is intended to be a **reasonable compromise** in most real-world cases



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.28

28

MFQ

- Extension of round robin
- Instead of only a single queue, MFQ has **multiple round robin queues**
 - Each queue has a **different** *priority level* and *time quanta*



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.29

29

Tasks and priorities

- Tasks at a higher priority **preempt** lower priority tasks
- Tasks at the same priority level are scheduled in a **round robin** fashion
- Higher priority tasks have **shorter** time quanta than lower priority tasks



COLORADO STATE UNIVERSITY

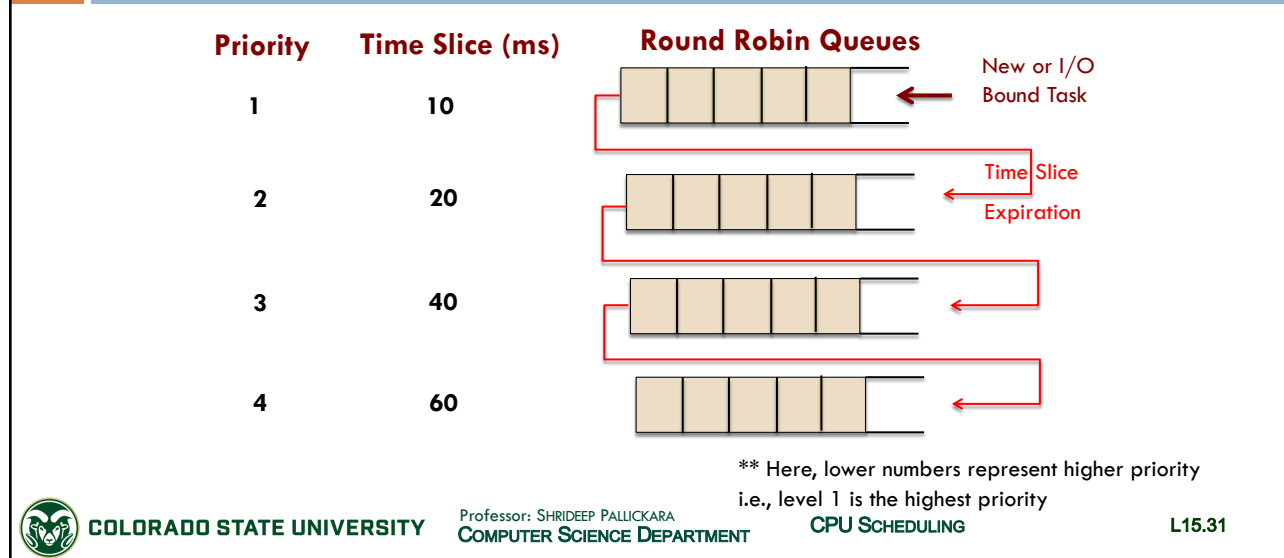
Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.30

30

MFQ: Example with 4 priority levels



31

Task movements and priority

- Tasks are moved between priority levels to **favor short tasks** over long ones
- Every time a task uses up its time quantum?
 - ▣ It **drops** a priority level
- Every time task yields the processor because it is waiting on I/O?
 - ▣ It **stays** at the same level, or is **bumped up** a level
- If the task completes ... it leaves

32

Impact on CPU and I/O bound processes

- A **new** CPU bound process will start as *high priority*
 - ▣ But it will quickly exhaust its time quantum and fall to the next lower priority, and then the next ...
- An I/O bound process with a modest amount of computing
 - ▣ Will always be **scheduled quickly**
 - Also, keeps the disk busy
- Compute bound tasks **run with a longer time quantum** to minimize switching overhead while sharing processor



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.33

33

Fair Share: Sharing the Microphone

- Imagine three speakers sharing one microphone, each entitled to the same speaking time
 - ▣ Over 60 minutes, each should get about 20 minutes
 - ▣ They can take turns of different lengths; what matters is total speaking time *over time*
 - If one speaker steps away, the others can use the microphone.
- The CPU works the same way: among 3 equally weighted processes that remain ready to run
 - ▣ Each should receive about $\frac{1}{3}$ of the CPU time
 - ▣ A process waiting for I/O lets others use the CPU



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.34

34

What about starvation and fairness?

- If there are too many I/O bound tasks, the compute bound tasks may receive no time on the processor
- MFQ scheduler **monitors every process** to ensure that it is receiving its fair share
 - ▣ For e.g., at each level, maintain two queues
 - Tasks whose processes have already reached their fair share are only scheduled if other processes at the same level have also received their fair share
- Periodically, processes not receiving their fair share have their tasks increased in priority
 - ▣ Tasks that receive more than their fair share have their priority reduced



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.35

35

Adjusting priority addresses strategic behavior

- From a selfish point of view, a task can keep its priority high by doing a short I/O request just before its quantum expires
 - ▣ With MFQ this will be detected, and its priority reduced to its fair level



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.36

36



37

Lottery scheduling

- Give processes **lottery tickets** for various system resources
 - ▣ E.g., CPU time
- When a scheduling decision has to be made
 - ▣ Lottery ticket is *chosen at random*
 - ▣ Process holding **ticket gets** the resource



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.38

38

Dealing with important processes: All processes are equal, but some processes are more equal than others

- More important processes are given **extra tickets**
 - ▣ Increase their odds of winning
- Let's say there are 100 outstanding tickets
 - ▣ 1 process holds 20 of these
 - ▣ Has 20% chance of winning each lottery
- A process holding a fraction f of tickets
 - ▣ Will get about a fraction f of the resource



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.39

39

Lottery Scheduling: Properties

[1/2]

- Highly **responsive**
 - ▣ Chance of winning is proportional to tickets
- Cooperating processes may **exchange** tickets
 - ▣ Process **A** sends request to **B**, and then hands **B** all its tickets for a faster response
- Avoids starvation
 - ▣ Each process holds at least one ticket Is guaranteed to have a non-zero probability of being scheduled



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.40

40

Lottery Scheduling: Properties

[2/2]

- Solves problems that are *difficult to handle* in other scheduling algorithms
- E.g., video server that is managing processes that feed video frames to clients
 - ▣ Clients need frames at 10, 20, and 25 frames/sec
 - ▣ Allocate processes 10, 20 and 25 tickets
 - CPU divided into approximately 10:20:25



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.41

41

LINUX COMPLETELY FAIR SCHEDULER (CFS)

Magicians protect their secrets not because the secrets are large and important, but because they are so small and trivial. The wonderful effects created on stage are often the result of a secret so absurd that the magician would be embarrassed to admit that that was how it was done.

Christopher Priest, The Prestige



42

Linux Completely Fair Scheduler (CFS)

- CFS accomplishes its proportional or fair-share goals in an alternate manner
 - ▣ Does so in a **highly efficient and scalable** fashion
- To achieve its efficiency goals, CFS aims to spend very little time making scheduling decisions through:
 - ▣ Its inherent design
 - ▣ Its clever use of data structures well-suited to the task



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.43

43

CFS: Basic Operation

- Whereas most schedulers are based around the concept of a fixed time slice, CFS operates a bit differently
- GOAL: Fairly divide a CPU evenly among all competing processes
 - ▣ Does so through a simple counting-based technique known as **virtual runtime (vruntime)**.



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.44

44

vruntime

- As each process runs, it **accumulates** vruntime
- In the most basic case, each process's vruntime increases at the same rate, in proportion with physical (real) time
- When a scheduling decision occurs, CFS will pick the process with the **lowest** vruntime to run next



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.45

45

How does the scheduler know when to stop the currently running process, and run the next one?

- Trade-off Space:
 - ▣ If CFS switches too often?
 - Fairness is increased: CFS will ensure that each process receives its share of CPU even over miniscule time windows
 - But at the cost of performance (too much context switching)
 - ▣ If CFS switches less often?
 - Performance is increased (reduced context switching)
 - But at the cost of near-term fairness



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.46

46

CFS manages this trade-off through various control parameters

□ `sched_latency`

- CFS uses this value to determine how long one process should run before considering a switch
 - Effectively determining its time slice but in a dynamic fashion

□ A typical `sched_latency` value is 48 (milliseconds)

- CFS divides this value by the number (n) of processes running on the CPU to determine the time slice for a process
 - And thus ensures that over this period of time, CFS will be completely fair



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.47

47

For example, if there are $n = 4$ processes running

- CFS divides the value of `sched_latency` by n to arrive at a per-process time slice of 12 ms
- CFS then schedules the first job and runs it until it has used 12 ms of (virtual) runtime
 - Then checks to see if there is a job with lower `vruntime` to run instead



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.48

48

But what if there are “too many” processes running?

- Wouldn't that lead to too small of a time slice, and thus too many context switches?
 - ▣ Yes!
- To address this issue, CFS adds another parameter, **min_granularity**, which is usually set to a value like 6 ms
 - ▣ CFS will never set the time slice of process to less than this value, ensuring that not too much time is spent in scheduling overhead



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.49

49

For example, if there are ten processes running

- Our original calculation would divide `sched_latency` by ten to determine the time slice (result: 4.8 ms)
 - ▣ However, because of min granularity, CFS will set the time slice of each process to 6 ms instead
- Although CFS won't (quite) be perfectly fair over the target scheduling latency (`sched_latency`) of 48 ms, it will be close
 - ▣ While still achieving high CPU efficiency



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.50

50

CFS utilizes a **periodic timer interrupt**

- CFS can only make decisions at **fixed time intervals**
- This interrupt goes off frequently (e.g., **every 1 ms**)
 - ▣ Giving CFS a chance to wake up and determine if the current job has reached the end of its run
- If a job has a time slice that is **not a perfect multiple** of the timer interrupt interval?
 - ▣ That is OK
 - ▣ CFS tracks `vruntime` precisely, which means that over the long haul, it will eventually approximate ideal sharing of the CPU



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.51

51

The contents of this slide-set are based on the following references

- Thomas Anderson and Michael Dahlin. *Operating Systems Principles and Practice*. 2nd Edition. ISBN: 978-0985673529. [Chapter 7]
- Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. 1st edition. CreateSpace Independent Publishing Platform. ISBN-13: 978-1985086593. [Chapter 9]
- Avi Silberschatz, Peter Galvin, Greg Gagne. *Operating Systems Concepts*, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330. [Chapter 6]
- Andrew S Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 2]



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

CPU SCHEDULING

L15.52

52