CS 330 - Operating Systems

Fair-share Scheduler

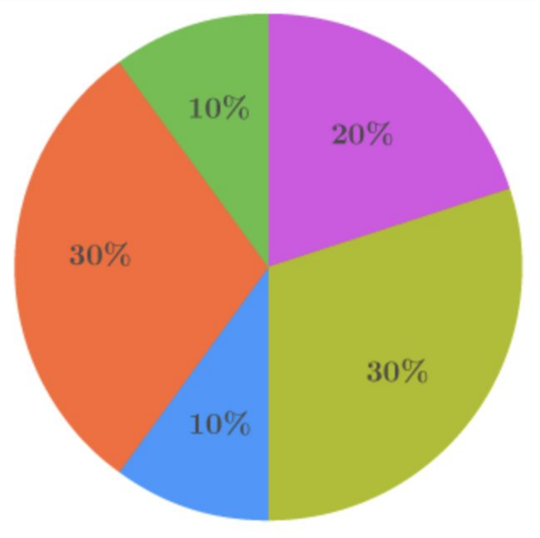
21-08-2025

Proportional-share Scheduling

- MLFQ compromises between TAT and RT
  - multiple queues of jobs with different priorities
  - “aging system” to shift batch style jobs to a lower priority
  - priority boosting to mitigate starvation
- But what if we wanted to emphasize fairness…

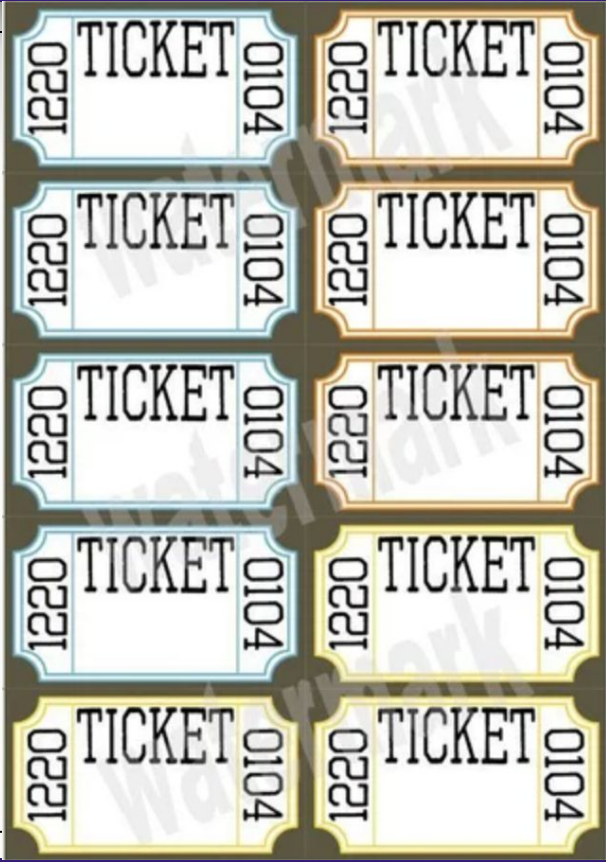
Proportional-share Scheduling

- Also referred to as a fair-share scheduler
- Focuses on trying to ensure that each job obtains a certain percentage of CPU time
- Approaches
    - Lottery Scheduling
    - Stride Scheduling
    - Completely Fair Scheduler (CFS)



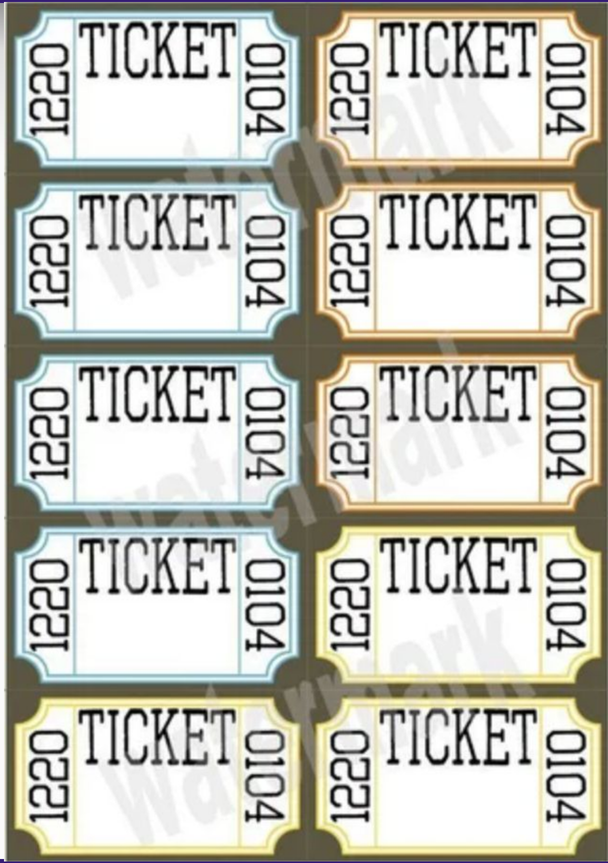
Lottery Scheduling

- Tickets represent job's share
- Three processes
  - P1 4 tickets
  - P2 3 tickets
  - P3 3 tickets
- Tickets represent the % share of CPU
  - P1 40%
  - P2 30%
  - P3 30%
- How to achieve this?



Lottery Scheduling

- Probabilistically
  - Scheduler knows total tickets (10)
  - Randomly picks a winning ticket (0-9)
    - **P1** **0-3**
    - **P2** **4-6**
    - **P3** **7-9**
  - Tickets determine which process to run
    e.g., — 3, 6, 9, 2, 4, 8, 0, 1, 5, 7



Lottery Scheduling

- Tickets distributed to process to indicate a share of resource
- E.g., CPU time, but could be used for other resources
- Winning numbers are chosen randomly by the scheduler
- Processes with more tickets are more likely to “win” and receive CPU time
- Effective light-weight approach that is probabilistically correct

Lottery Scheduling Example

- What is the likely-hood of running each job:
  - A = 10%
  - B = 20%
  - C = 70%
- Scheduler picks:
  - 99, 16, 80, 60, 13, 45, 6, 56, 76, 82, 40, 5, 27, 88, 7

Lottery Scheduling Example

- What is the likely-hood of running each job:
  - A = 10%
  - B = 20%
  - C = 70%
- Scheduler picks:
  - 99, 16, 80, 60, 13, 45, 6, 56, 76, 82, 40, 5, 27, 88, 7
  - C, B, C, C, B, C, A, C, C, C, C, A, B, B, A
- Observed scheduling results?
  - A = ~20%
  - B = ~27%
  - C = ~53%

Ticket Manipulation

- Ticket Currency
  - The scheduler provides a set number of tickets to each user
  - This represents a global currency
  - A user can allocate tickets to its tasks and in arbitrary numbers
  - The scheduler will scale ticket distribution to the number of tickets provided to that job in the global currency
- User A — 100 tickets
- User B — 100 tickets

Ticket Manipulation

- Ticket Currency
  - The scheduler provides a set number of tickets to each user
  - This represents a global currency
  - A user can allocate tickets to its tasks and in arbitrary numbers
  - The scheduler will scale ticket distribution to the number of tickets provided to that job in the global currency
- User A — 100 tickets
  - Task A1 — 500 tickets
  - Task A2 — 500 tickets
- User B — 100 tickets
  - Task B1 — 10 tickets

Ticket Manipulation

- Ticket Currency
  - The scheduler provides a set number of tickets to each user
  - This represents a global currency
  - A user can allocate tickets to its tasks and in arbitrary numbers
  - The scheduler will scale ticket distribution to the number of tickets provided to that job in the global currency
- Task A1 → 500 (A's currency) → 50 (global currency)
- Task A2 → 500 (A's currency) → 50 (global currency)
- Task B1 → 10 (B's currency) → 100 (global currency)

Ticket Manipulation

- Ticket Transfer
  - Sometimes a job does not need all the tickets it is provided
  - Allows a job to temporarily donate its tickets to another job
  - Can be useful when a job relies on the results of another
    - Helps maximize resource allocation to get the result faster
- Tickets are returned to loaner after the job completes

Ticket Manipulation

- Ticket Inflation
  - A process can temporarily increase or decrease the number of tickets it has (global currency)
  - Doesn't require communication between processes like ticket transfer
  - But works only in a cooperative setting as a greedy process could starve others

Lottery Scheduling Issues

- Requires many time slices before ideal “fairness” is reached
- How should we assign tickets?

Stride Scheduling

- Attempt to reach a more optimal fairness outcome over shorter time slices by limiting randomness
- Calculate a stride for each job by taking its number of tickets and dividing it by a very large number
- As the processes run, we add their stride value to a counter associated with the process (call the pass value)
- The scheduler always selects the job with the lowest pass value to run
- If there is a tie, one may be chosen randomly

Stride Scheduling Example

- Suppose A, B, and C, with 100, 50, and 250 tickets
- Divide it by a large number 10000
- A - stride=100
- B - stride=200
- C - stride=40

Stride Scheduling Example

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C

Stride Scheduling Issues

- Global state for process
  - New job enters in the middle
  - What should its pass value be? Should it be set to 0?
- With lottery scheduling, there is no global state
  - Add a new process with whatever tickets it has, update the single global variable to track how many total tickets we have, and continue

Linux Scheduler

- Completely Fair Scheduler (CFS)
  - Highly efficient and scalable fair-share scheduler
- Keeps track of the amount of time a process has run on the CPU with virtual time (vruntime)
- Picks the process with the lowest vruntime to run next
- Uses two control parameters:
  - sched_latency – how long should a process run before switching
    - Divided by the number of jobs to determine slice time per job
  - min_granularity – the smallest possible time slice for a process
- Utilizes a timer interrupt to frequently wake up and see if a switch is necessary

Completely Fair Scheduler

- Suppose
  sched_latency = 48 ms and
  min_granularity = 6 ms
- 4 processes running
  - per-process time slice of 12 ms
- CFS schedules the first job and runs it until 12 ms
- Then checks if there is a job with lower vruntime to run
  - CFS would switch to one of the three other jobs, and so forth.
- If two of them complete, the remaining two run for 24 ms
- What if there are 10 processes?
- What about priority?

Completely Fair Scheduler

- CFS supports priority scaling as well via a nice level
- Nice values range from -20 to +19 where positive values imply lower priority
- Constants represents the weight to be applied to the sched_latency (default is 1024)
    - Weight proportion is calculated as the current job weight over the sum of weights for all job

Completely Fair Scheduler

- Jobs A and B are in the system
  - Assume a sched_latency of 48ms
- Job A is given a priority of -3 (1991)
- Job B is given a priority of 0 (1024)
- What are the time slices:
  - Time Slice for job A = (1991/ (1991+1024)) * 48 = 32ms
  - Time Slice for job B = (1024 / (1991+1024)) * 48 = 16ms

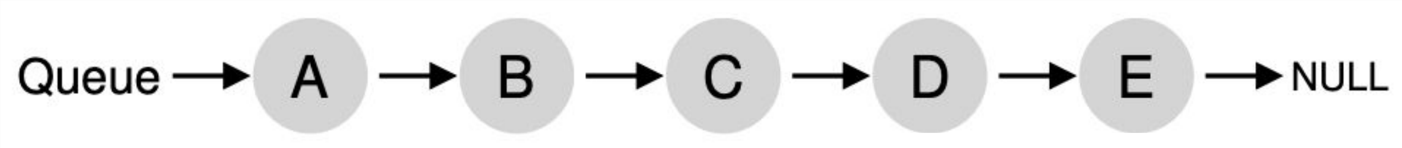
Multiprocessor

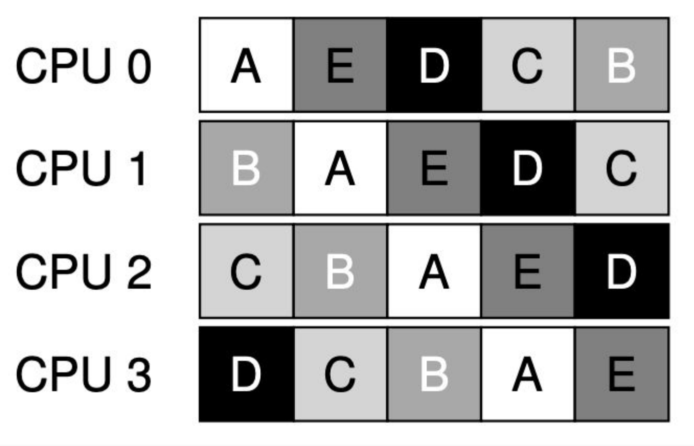
Multiprocessor Scheduling

- Different from single-processor
  - Contains multiple CPUs
  - Multiple caches
  - Data sharing across multiple processors
- Issues due to caches
  - Data cached in CPU 1 may be required in CPU 2 due to scheduling pattern
    - Cache coherence
    - Locality - temporal and spatial - affected
  - Cache affinity

Multiprocessor Scheduling

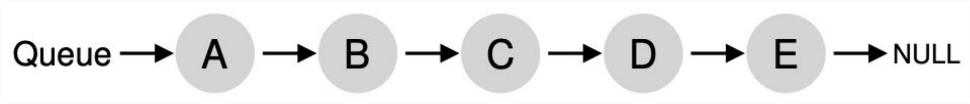
- Single queue
  - Simple

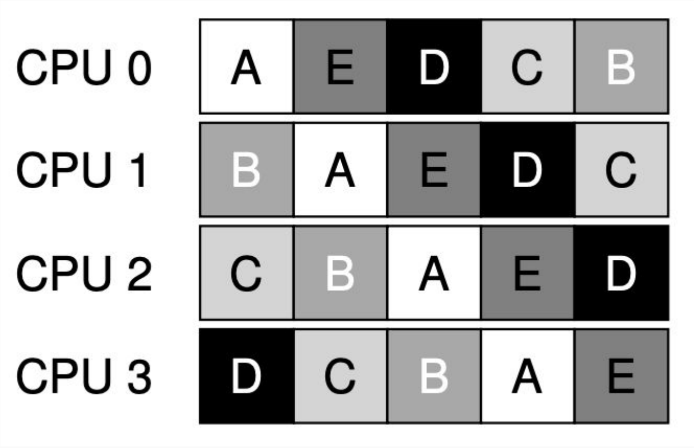




Multiprocessor Scheduling

- Single queue
  - Simple
  - Shortcomings?
    - Scalability - multiple processors cannot access the queue at the same time
    - Cache affinity may be affected





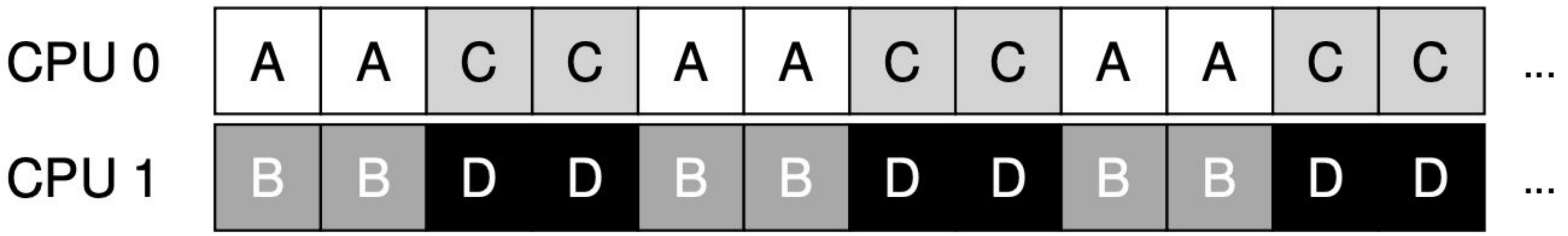
Multiprocessor Scheduling

- Single queue
  - Simple
  - Shortcomings
    - Scalability
    - Cache affinity

- Multiple queues
  - Multiple scheduling queues follow their own scheduling algo.
  - OS decides which CPU to schedule on
  - Scalable with cache affinity

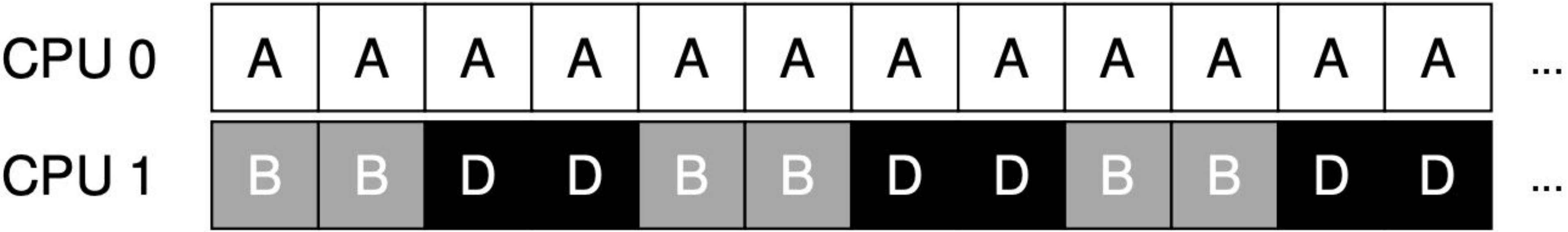
Multiprocessor Scheduling

- Multiple queues
  - Multiple scheduling queues follow their own scheduling algo.
  - OS decides which CPU to schedule on
  - Scalable with cache affinity



Load Imbalance and Migration

- Suppose job C finishes

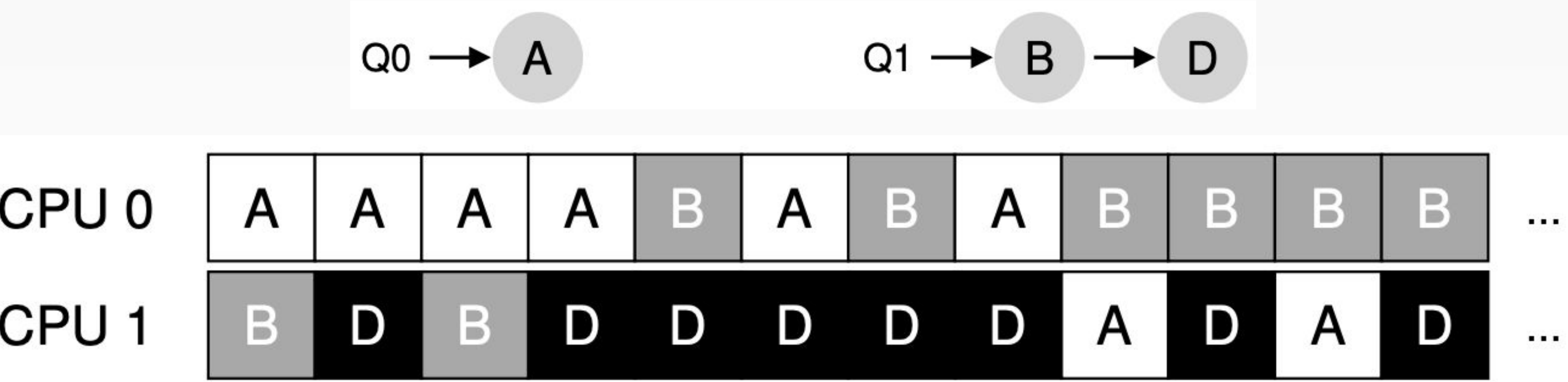


Load Imbalance and Migration

- Or both job A & C finish

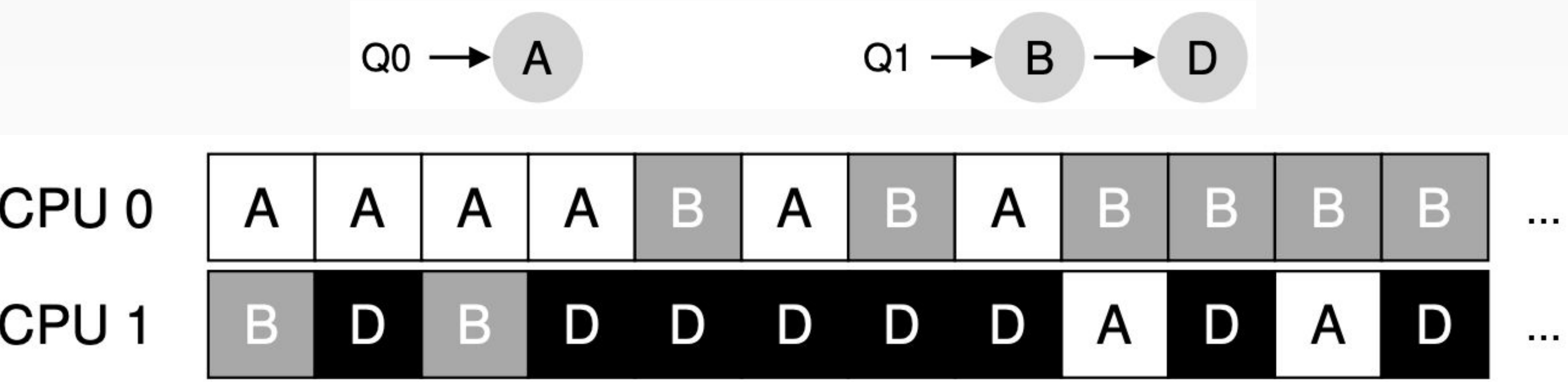
Load Imbalance and Migration

- Switch jobs occasionally



Load Imbalance and Migration

- Switch jobs occasionally



- How to migrate?
  - Work stealing