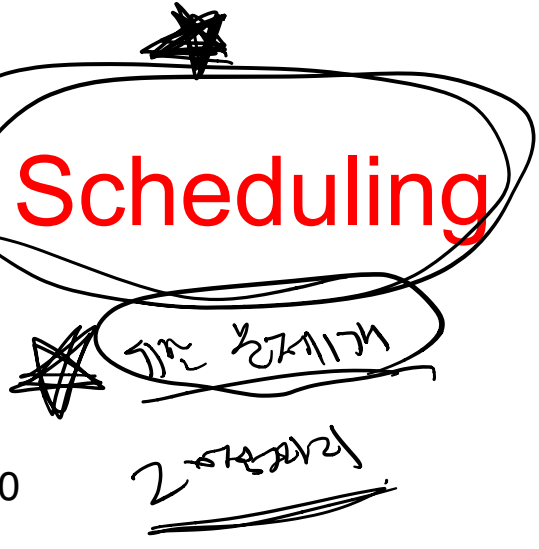
Lecture Note 3. Scheduling



March 20, 2020

Jongmoo Choi

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~choijm

Contents

From Chap 7~11 of the OSTEP

Chap 7. Scheduling: Introduction

- ✓ Workload assumptions and Scheduling Metrics
- ✓ Algorithms: FIFO, SJF, STCF, RR
- ✓ Incorporating I/O

Chap 8. Scheduling: MLFQ (Multi-Level Feedback Queue)

- ✓ Basic rules
- ✓ Attempts: Change priority, Boost priority, Better accounting
- ✓ Tuning MLFQ and other issues

Chap 9. Scheduling: Proportional Share

- ✓ Basic concept: Lottery, Stride
- ✓ Ticket mechanism, implementation, example and issues

Chap 10. Multiprocessor Scheduling

- ✓ Background: load balancing, cache affinity
- ✓ Scheduling: single queue, multi-queue

Chap 11. Summary Dialogue on CPU virtualization

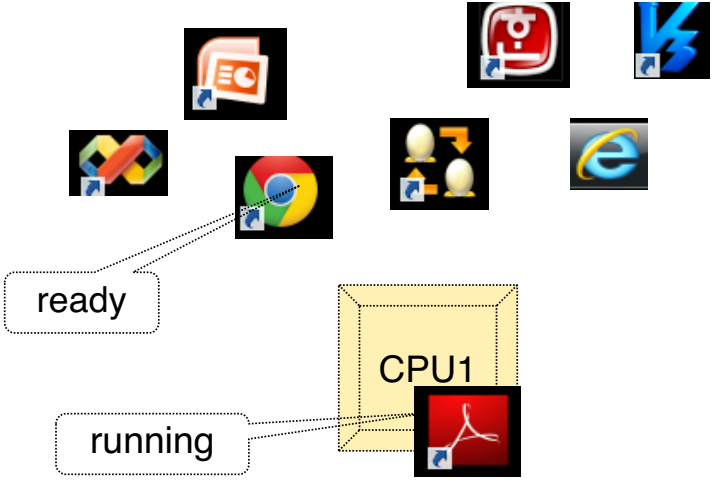
Chap 7. Scheduling: Introduction

Scheduling

- ✓ Multiple actors want to use (limited) resources at a time
- ✓ Make order to select actors who can use the resources

Process Scheduling

- ✓ Actor: process, Resource: processor (CPU)
- ✓ Select a process who run on a processor (or processors)



7.1 Workload assumption

Workload

- The amount of work to be done (dictionary)
- How much resources are required by a set of processes with the consideration of their characteristics (in computer science)

A simple assumption about processes (also called as job in the scheduling research area)

- Each job runs for the same amount of time
- All jobs arrives at the same time
- Once started, each job runs to completion
- All jobs only use the CPU (no I/O)
- The run-time of each job is known in advance

- c.f.) unrealistic, but we will relax them as we go

7.2 Scheduling Metrics

Metrics

- Something that we use to measure (e.g. performance, reliability, …)

Metrics for scheduling

- Turnaround time
  - Tturnaround = Tcompletion - Tarrival
- Response time
  - Tresponse = Tfirstrun - Tarrival
- Fairness
  - E.g.) Tcompletion of P1 vs. that of P2
- Throughput
  - E.g.) number of completed processes / 1 hour
- Deadline
  - E.g.) Tturnaround < Tdeadline
- ….

? What do you think first when we choose a restaurant for lunch? (among above)

? What does the owner of the restaurant think first?

7.3 FIFO (First In, First Out)

FIFO

- ✓ Schedule a process that arrives first (a.k.a FCFS (First Come First Serve))
- ✓ Example

  1) three processes: A, B, C, 2) run-time: 10 seconds, 3) arrival time: 0s (tie-break rule: alphabet in this example)

  What is the average turnaround time?
- ✓ Another exa

  1) three proces

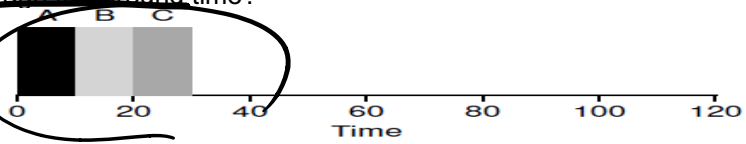


Figure 7.1: FIFO Simple Example

Now, what is the average turnaround time?

turnaround time,
response time.
Average turnaround time 계산하는 법

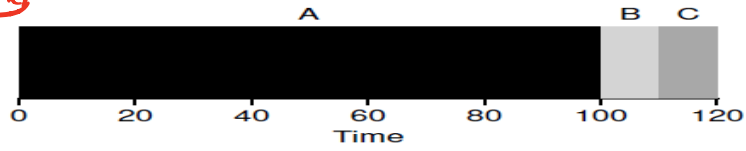


Figure 7.2: Why FIFO Is Not That Great

7.3 FIFO (First In, First Out)

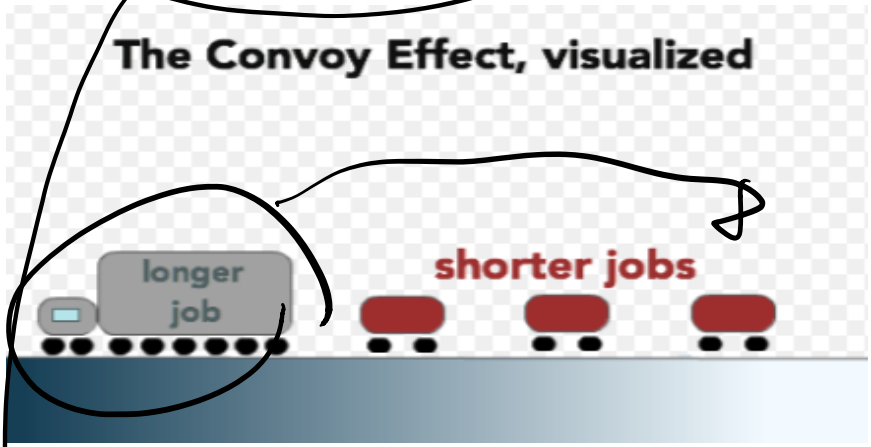
FIFO

Pros)

1) Clearly simple, 2) Easy to implement

Cons)

1) May cause a long waiting time (known as convoy effect)



(Source: http://web.cs.ucla.edu/classes/fall14/cs111/scribe/7a/index.html)

How can we overcome this long waiting?

7.4 SJF (Shortest Job First)

SJF

- ✓ Give a higher priority to the shortest job (a.k.a Shortest Process Next (SPN))
  - "ten-items-or-less" in a grocery store
- ✓ Revisit the previous example again
  - 1) three processes: A, B, C, 2) run-time: 100s for A, 10s for B and C

What is the a

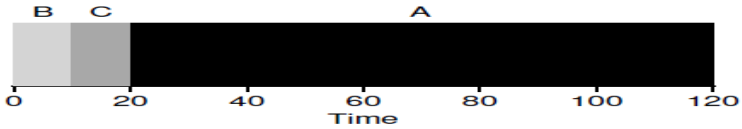


Figure 7.3: SJF Simple Example

- ✓ Pros)

Proved as an

- ✓ Cons)

What if B and C arrive a little bit late than A? (e.g. assume 10, not 0)

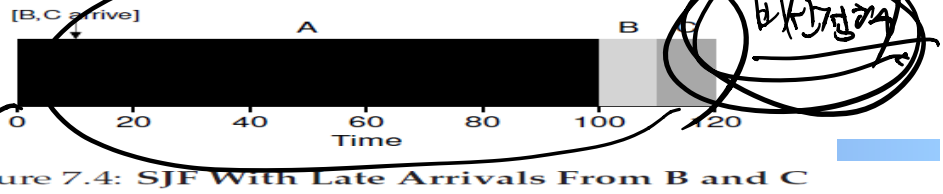
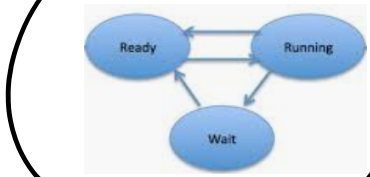


Figure 7.4: SJF With Late Arrivals From B and C

7.5 STCF (Shortest Time-to-Completion First)

STCF

- ✓ Similar to SJF, but preemptive version (a.k.a Shortest Remaining-Time next (SRT))
- ✓ 1) Non-preemptive scheduling
  - Run a job to completion
- ✓ 2) Preemptive scheduling
  - Can stop a job (even though it is not completed yet) to run another job
  - All modern schedulers are preemptive
  - Require the context switch
- ✓ Example
  - 1) three processes: A, B, C, 2) run-time: 100s for A, 10s for B and C, 3) arrival time: 0s for A, 10s for B and C.



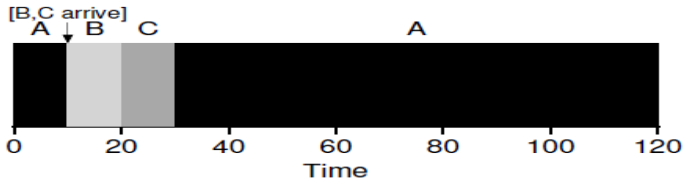


Figure 7.5: STCF Simple Example

vs

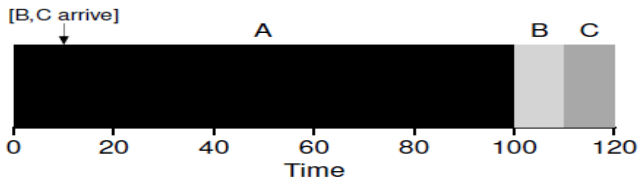


Figure 7.4: SJF With Late Arrivals From B and C

7.6 Response time

Turnaround time

- ✓ A good metric for a batching system

Response time

- ✓ More important for an interactive system?

  User would sit at a terminal, working something interactively (e.g. move a mouse, type in a letter, visit a site, and so on)

Revisit the example with SJF (also FIFO)

- ✓ 1) three processes: A, B, C, 2) run-time: 5 seconds, 3) arrival time: 0s (tie-break rule: alphabet in this example)
- ✓ What is the average turnaround time?
- ✓ How

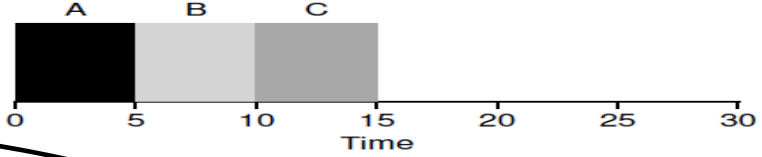


Figure 7.6: SJF Again (Bad for Response Time)

FIFO
SJF
STCF
→ Turnaround time 최적 (batch processing)

? Imagine that you move a mouse and wait for a 5s.

7.7 RR (Round-robin)

RR

- Instead of running a job to completion, it runs a job for a time slice (sometimes called a scheduling quantum) and switch to the next job in the run queue
- Repeatedly switch jobs until jobs are finished
- Example

  1) three processes: A, B, C, 2) run-time: 5s, 3) arrival time: 0s (same to the previous slide)
  RR with time slice = 1s (different here: non-preemptive in the previous slide)
  What is the average response time?
  What is the average turnaround time?

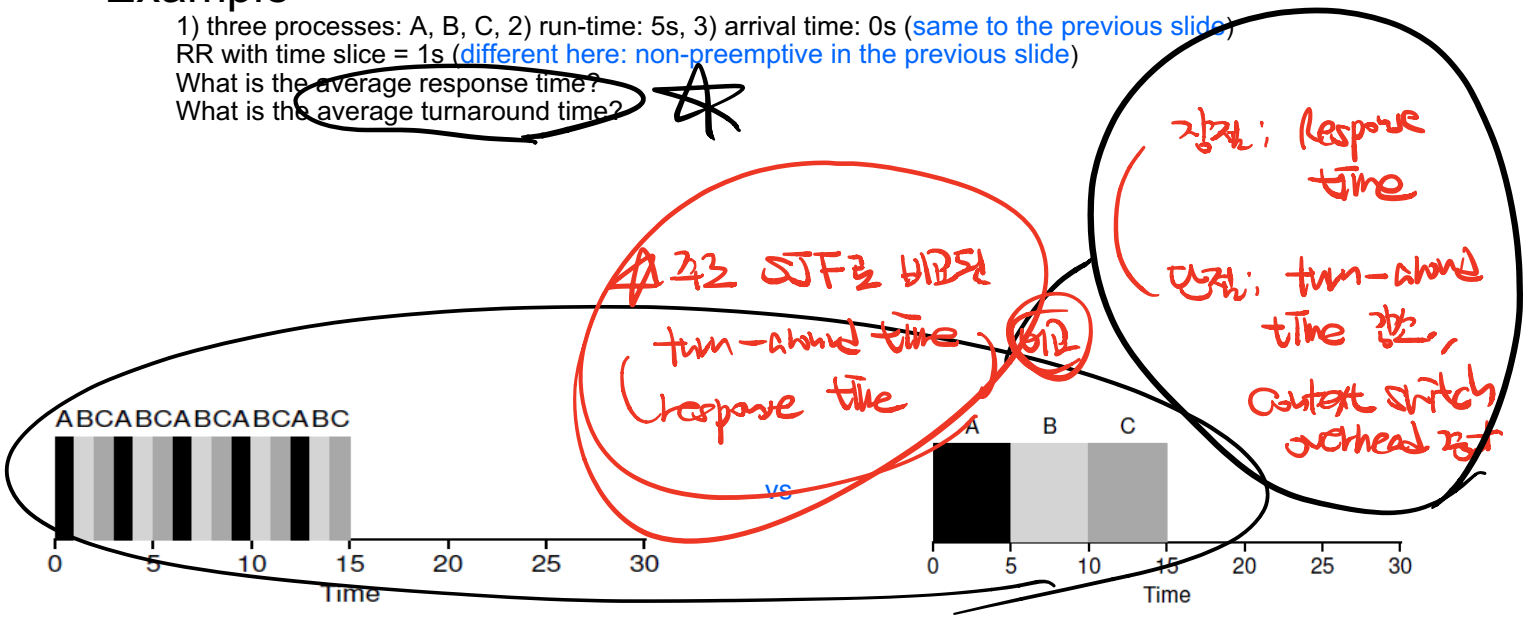


Figure 7.7: **Round Robin (Good for Response Time)**

Figure 7.6: **SJF Again (Bad for Response Time)**

? What if the time slice is set as 500ms or 100ms or 10ms. Discuss tradeoff

7.7 RR (Round-robin)

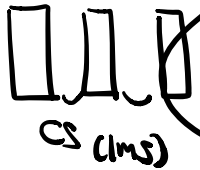
Tradeoff of time quantum

- Small: good responsiveness, high context switch overhead
- Large: low context switch overhead, bad responsiveness
- We need to balance the tradeoff
  - Good response time with reasonable overhead
  - E.g. time quantum: 10ms, context switch overhead: 1ms

Tra

> TIP: AMORTIZATION CAN REDUCE COSTS
>
> The general technique of **amortization** is commonly used in systems when there is a fixed cost to some operation. By incurring that cost less often (i.e., by performing the operation fewer times), the total cost to the system is reduced. For example, if the time slice is set to 10 ms, and the context-switch cost is 1 ms, roughly 10% of time is spent context switching and is thus wasted. If we want to *amortize* this cost, we can increase the time slice, e.g., to 100 ms. In this case, less than 1% of time is spent context switching, and thus the cost of time-slicing has been amortized.

You can not have your cake and eat it too.

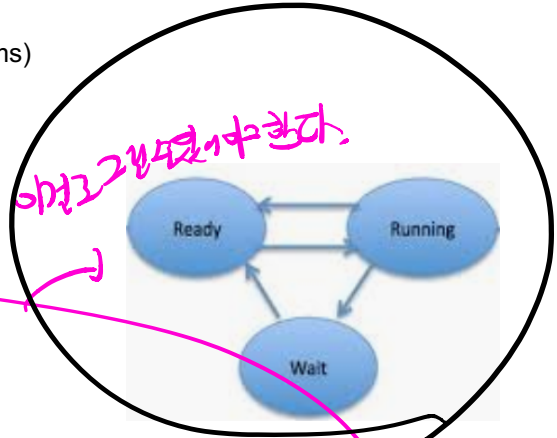


- About the question, "explain which process you prefer to schedule when there are two processes, browser and synchronizer (backup apps)" ? Considerations: 1) interactive or batch, 2) importance, 3) fairness, 4) real-time, …

7.8 Incorporating I/O

Most of applications do I/Os

- ✓ Example
  - Two jobs A and B, both need 50ms of CPU time
  - A runs for 10 ms and then issue an I/O request (it takes 10 ms)
- ✓ What to do while performing I/Os?
  - Busy waiting: Figure 7.8
  - Blocked: Figure 7.9
- ✓ How to implement the Figure 7.9



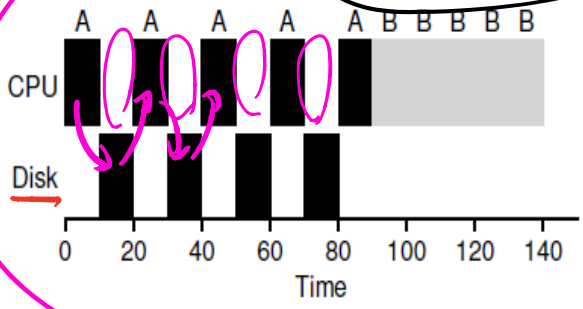


Figure 7.8: Poor Use of Resources

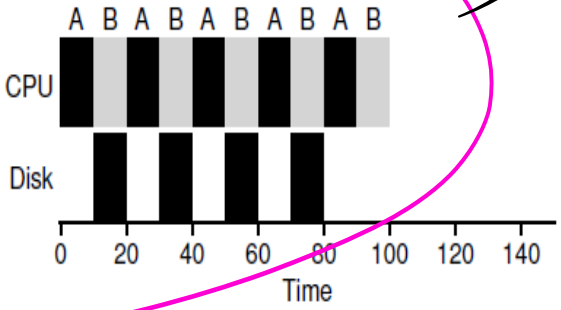


Figure 7.9: Overlap Allows Better Use of Resources

7.9 No More Oracle

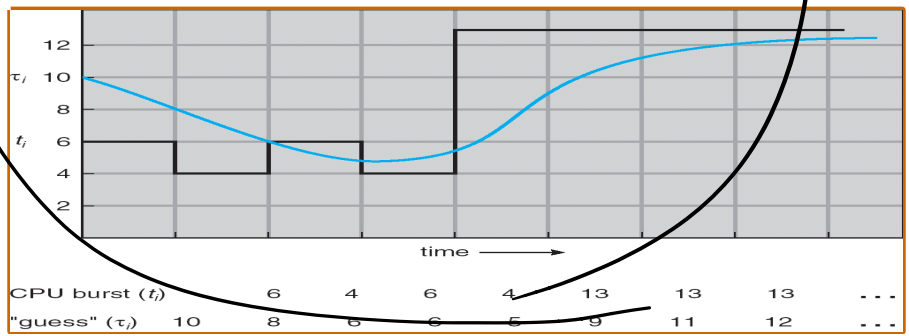
How to predict the length of a job (run time)?

- ✓ By user specification
- ✓ By prediction (approximation)

The CPU time length will be similar in length to the previous ones (characteristics of program behavior) ? exponential moving average

Validation with ? =0.5 and ?0=10 (? determines the weight of each history)

where



- 6th Simple question to take attendance: In real life, a good example of SJF is “ten-items-or-less” in a grocery store. Then what is an example of RR in real life? (until 6 PM, April 2nd)

8. MLFQ

Existing scheduling policies

- ✓ FIFO (6 page), SJF (8 page), STCF(9 page): good for turnaround time, terrible for response time
- ✓ RR (11 page): vice versa

How to optimize the turnaround time while minimizing response time?

- MLFQ (Multi-Level Feedback Queue)
  - By F. Corbato (Turing Award Winner)
  - Approach: Learn from the past to predict the future

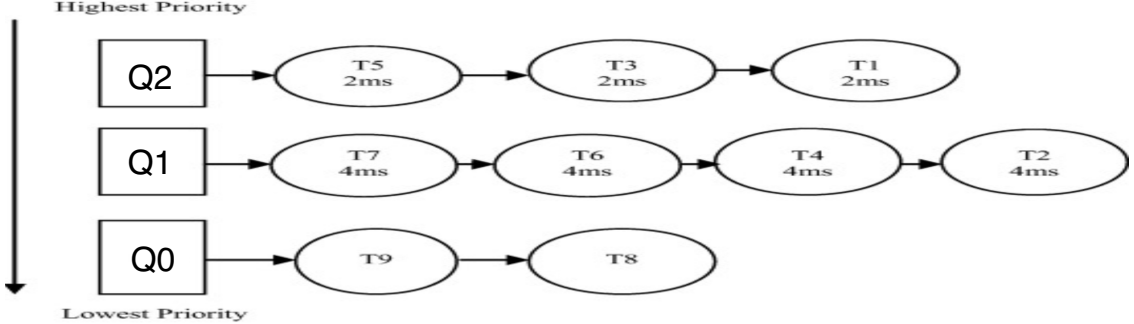
> TIP: LEARN FROM HISTORY
>
> The multi-level feedback queue is an excellent example of a system that learns from the past to predict the future. Such approaches are common in operating systems (and many other places in Computer Science, including hardware branch predictors and caching algorithms). Such approaches work when jobs have phases of behavior and are thus predictable; of course, one must be careful with such techniques, as they can easily be wrong and drive a system to make worse decisions than they would have with no knowledge at all.

8.1 MLFQ: Basic Rules

MLFQ

- ✓ Consist of multiple queues
- ✓ Each queue is assigned a different priority level
- ✓ A job that is ready to run is on a single queue (running or blocked jobs are out of the queues)
- ✓ A job with higher priority (a job on a higher queue) is chosen to run next (RR among jobs in the same queue)

- **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) = Priority(B), A & B run in RR.



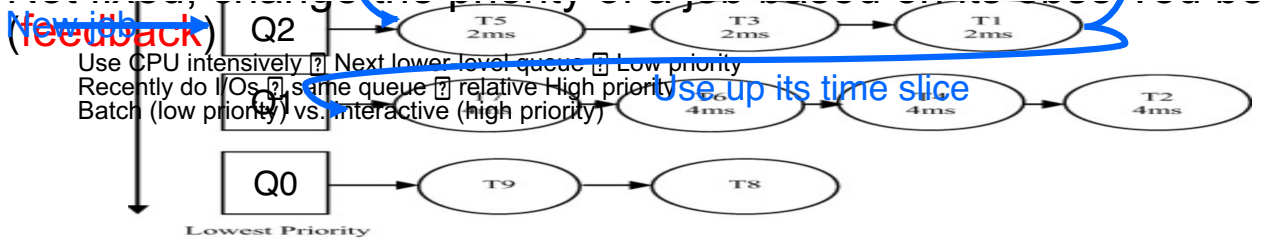
8.2 Attempt #1: How to Change Priority

How to assign a priority to each process?

- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4a:** If a job uses up an entire time slice while running, its priority is *reduced* (i.e., it moves down one queue).
- **Rule 4b:** If a job gives up the CPU before the time slice is up, it stays at the *same* priority level.

Give up before spending its whole time slice

...avior



8.2 Attempt #1: How to Change Priority

Examples

- ✓ Example 1: A Single Long-Running Job ? Fig. 8.2
  Assumption: Three queues (Q2, Q1, Q0), one job, 10ms time slice
- ✓ Example 2: A long and a new arrived Job ? Fig. 8.3
  Just arrived job ? MLFQ presumes the job is a short job ? Give high priority
  - Really a short job: run quickly and complete (approximates SJF)
  - If not: move down the queues, proving itself as a long-running
- ✓ Example 3: What about I/O? ? Fig. 8.4
  Assumption: two jobs, A: long-running job, B: short-intensive job
  MLFQ keep a ~~process at the~~ same queue if it gives up CPU before using up its time slice (rule 4b)
  - Prefer I/O intensive job for good response time

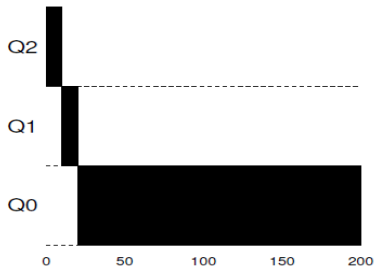


Figure 8.2: **Long-running Job Over Time**

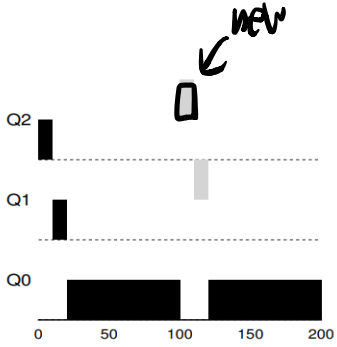


Figure 8.3: **Along Came An Interactive Job**

Figure 8.4: **A Mixed I/O-intensive and CPU-intensive Workload**

8.2 Attempt #1: How to Change Priority

Problem with our current MLFQ

- ✓ Pros of the current version
  - Share CPU fairly among long-running jobs
  - Allow short-running or I/O intensive jobs to run quickly
- ✓ Issues
  - Starvation
    - If there are "too many" interactive jobs, long-running jobs will never receive any CPU time (they starve)
  - User can trick the scheduler (game the scheduler)
    - Just before the time slice over, issue an I/O request ? remain in the same queue unfairly
  - A program may change its behavior
    - CPU-intensive at the first phase ? interactive at the later phase (e.g. service user request after long initialization)

8.3 Attempt #2: The Priority Boost

New rule for avoid starvation

- ✓ One approach: periodic boosting
- ✓ Ex

- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

Three jobs, two interactive jobs and one long-running job
Priority boost every 50 ms

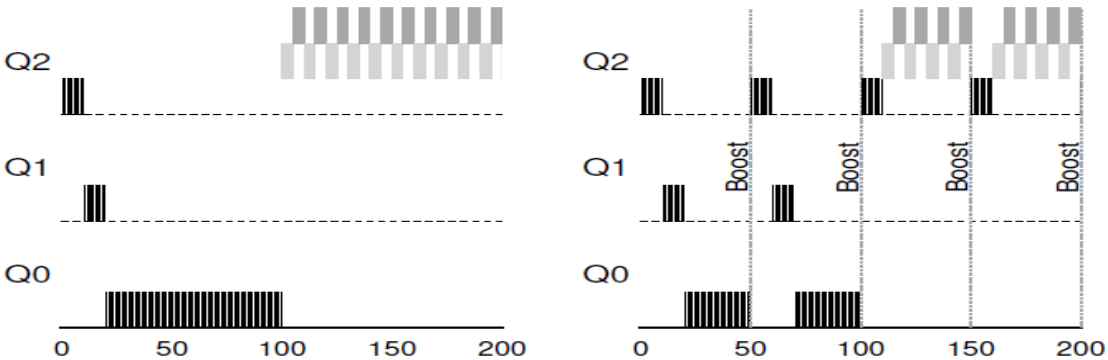


Figure 8.5: **Without (Left) and With (Right) Priority Boost**

8.4 Attempt #3: Better Accounting

How to prevent gaming of MLFQ scheduler?

- ✓ Change the rule 4a and 4b [?] instead of forgetting how much of a time slice a job used at a given queue, keep track it. Once a job has used its allotment, it is demoted to the next queue

- **Rule 4:** ~~Once a job uses up its time allotment at a given level~~ (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).

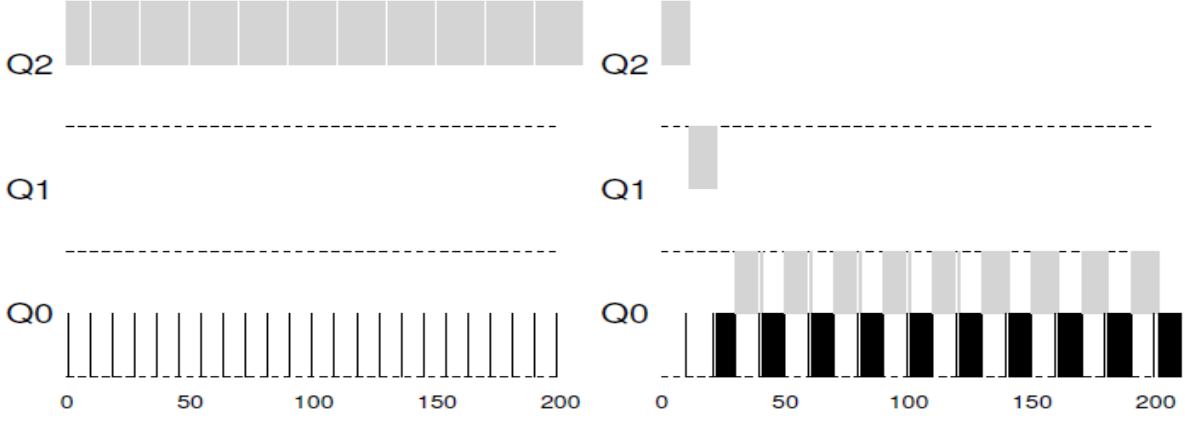


Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

8.5 Tuning MLFQ and Other Issues

Parameters

- ✓ Issues
  - How many queues?
  - How big should the time slice be per queue? Same or Different?
  - How often do the priority boost?
- ✓ Many MLFQ variants with diverse parameter settings
  - Different time slice per queue: shorter for higher priority queue and vice versa (10, 20 and 40ms in Fig. 8.7 ? can reduce context switch overhead)
  - Solaris case: Table based
  - BSD, Linux: Decay based (mathematical)
  - Support user advice (e.g. nice system call)

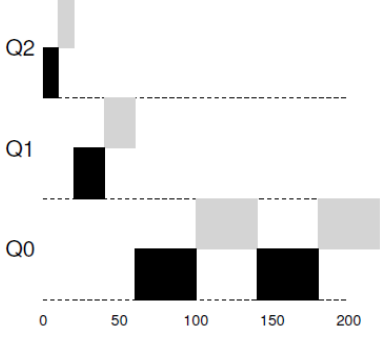


Figure 8.7: **Lower Priority, Longer Quanta**

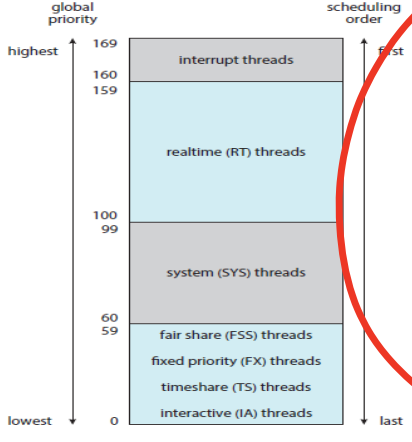


Figure 6.24 Solaris scheduling

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

(Source: A. Silberschatz, "Operating System Concept")

8.6 MLFQ: Summary

Name analysis

- ✓ Multi-level: multiple queues
- ✓ Feedback: based on history (track job's behavior over time and treat them accordingly)

Final rules

- **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) = Priority(B), A & B run in RR.
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

Feat

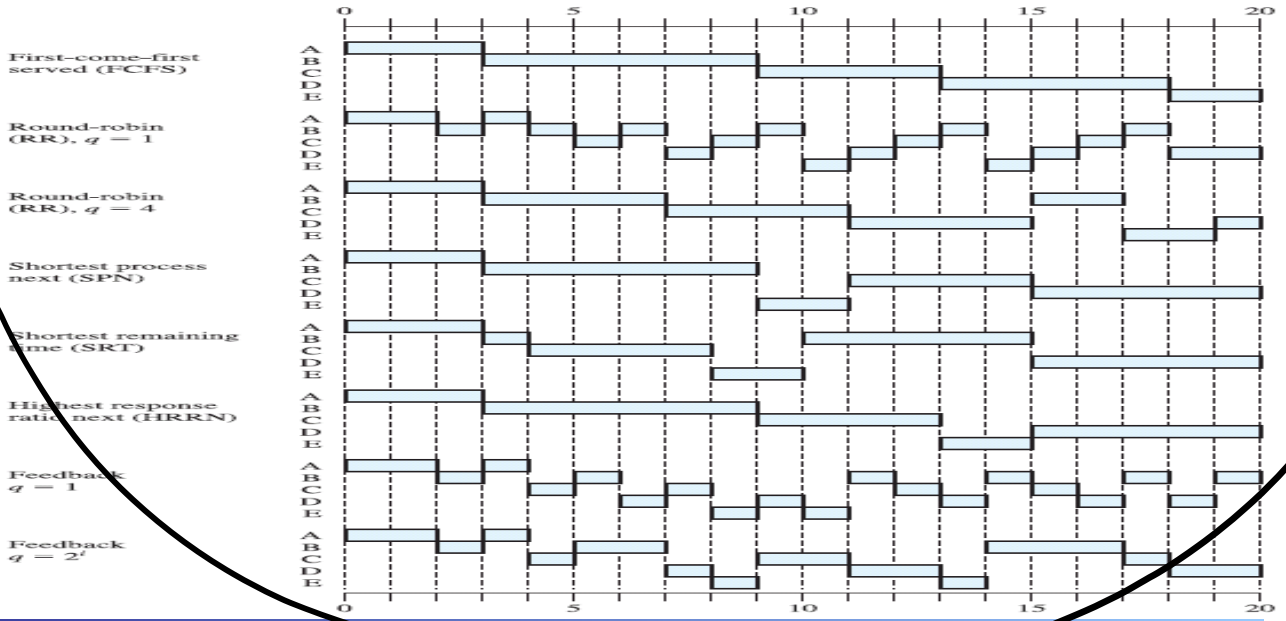
- ✓ rm

8.6 Scheduling Comparison

Workload: 5 processes (jobs)

Scheduli

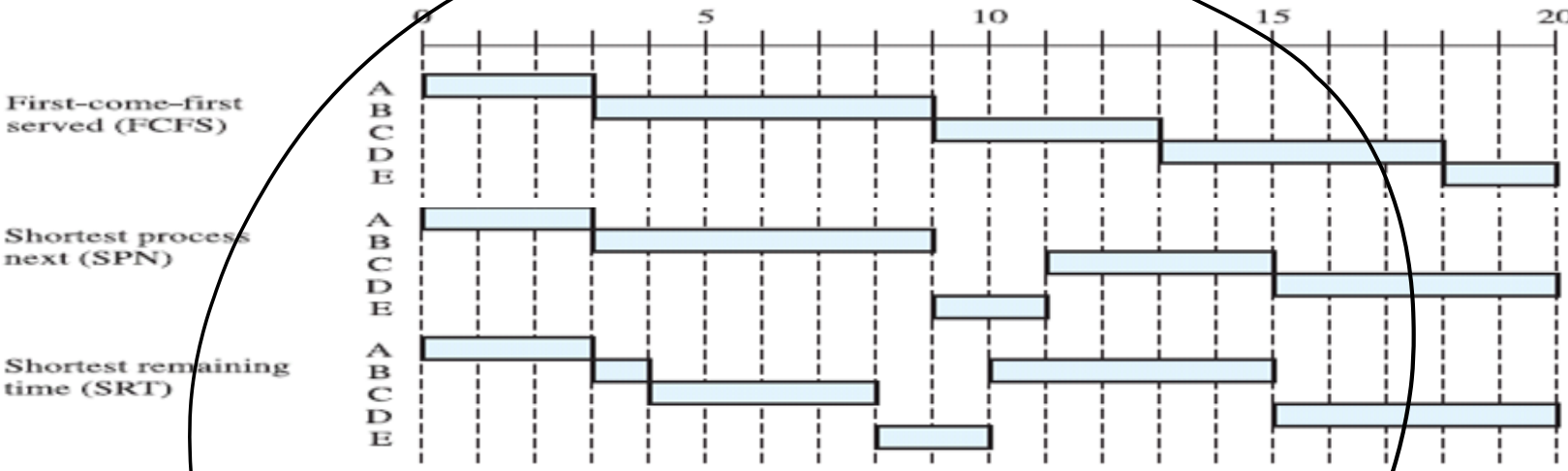
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



(Source: "Operating systems: Internals and Design Principle" by W. Stalling)

8.6 MLFQ: Summary

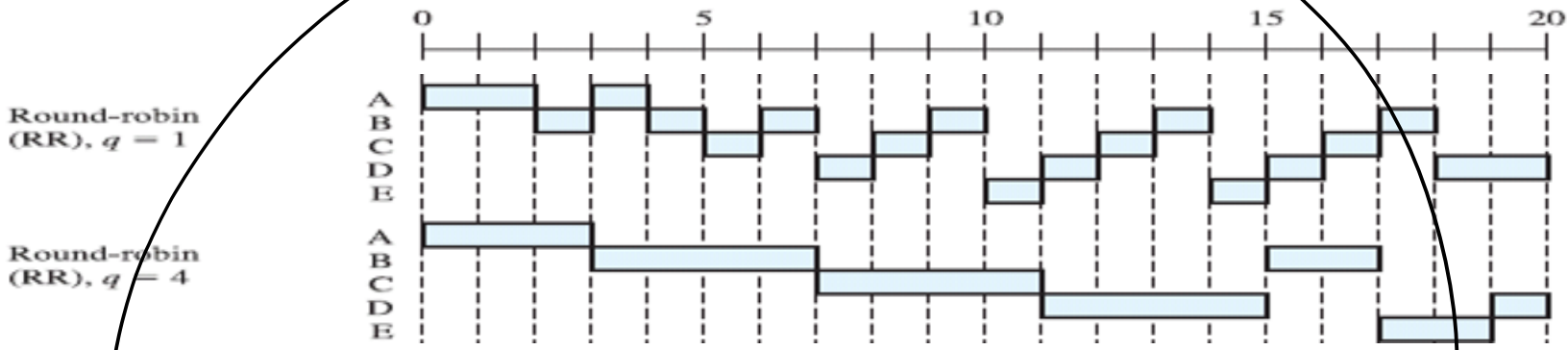
Example: FCFS(FIFO), SPN (SJF), SRT (STCF)



Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

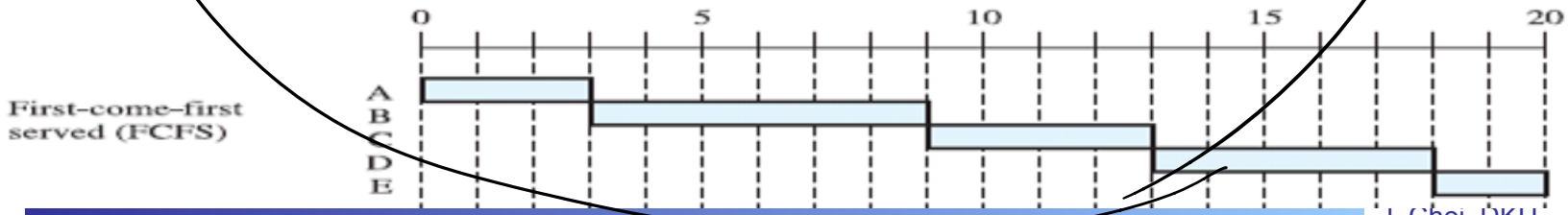
8.6 MLFQ: Summary

Example: RR (time quantum = 1), RR (time quantum = 4)



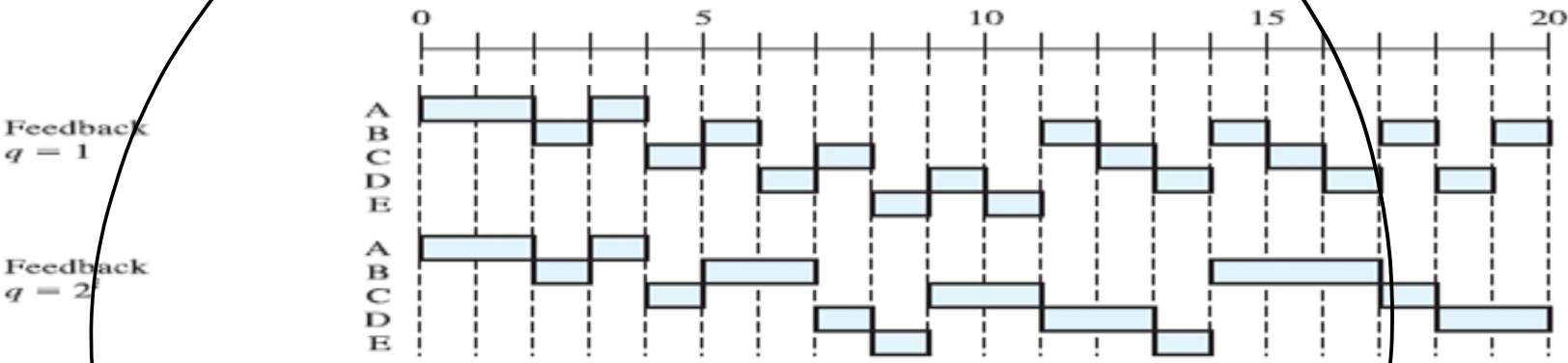
Process	Arrival Time	Service Time
A	0	3
B	2	6

- Note: In our video clip, I said that the average response time of RR is "(0+0+0+1+2)/5". But, it is my mistake. The real answer is "(0+0+1+1+2)/5". Thank you for the student who reported this to me :-)



8.6 MLFQ: Summary

Example: MLFQ (time quantum = 1), MLFQ (time quantum = 1, 2, 4, 8, …)



Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- 7th Simple question to take attendance: What is the HRRN (Highest Response Ratio Next) shown in the page 24? (until 6 PM, April 8st)

lottery, stride

Proportional Share (fair share)

- ✓ Concept: instead of turnaround time or response time, it tries to guarantee that each job obtain a certain percentage of CPU time (especially important for Cloud system)
- ✓ Scheduling algorithms: Lottery, Stride, …

Example: 3 VMs A, B, C with 3 : 2 : 1 share ratio

A	2	4	4	6	8	8	8	10	10
B	3	3	6	6	6	9	9	9	12
C	6	6	6	6	6	6	12	12	12

9.1 Basic Concept: Tickets Represent Your Share

Lottery scheduling

- ✓ Made by Waldspurger and Weihl
- ✓ Schedule a job who wins the lottery
- ✓ A job that has more tickets has more chance to win
  - Ticket: represent the share of a resource
  - Two jobs, A has 75% tickets while B has 25% tickets ? win probability with 75% and 25% ? 75% of CPU is expected to be used by A
- ✓ Example
  - Total tickets: 0~99, A: 0~74, B: 75~99

80% for A, 20% for B in this example (since it is based on probability). But, the longer it runs, the more lik

Here is an example output of a lottery scheduler's winning tickets:

```
63 85 70 39 76 17 29 41 36 39 10 99 68 83 63 62 43  0 49 49
```

Here is the resulting schedule:

```
A     A  A     A  A  A  A  A  A     A     A  A  A  A  A  A
   B        B                    B     B
```

9.2 Ticket Mechanisms

Ticket currency

- ✓ Allow users to allocate tickets among their own jobs with correct global value
- ✓ Example
  - Two users, A: 100 tickets, B: 100 tickets
  - A has two jobs. A gives them each 500 tickets
  - B has only one job. B gives it 10 tickets
  - How many tickets are given into three jobs with a global viewpoint?

Ticket transfer

- ✓ A job t
- ✓ Espec

```
User A -> 500 (A's currency) to A1 ->  50 (global currency)
       -> 500 (A's currency) to A2 ->  50 (global currency)
User B ->  10 (B's currency) to B1 -> 100 (global currency)
```

Ticket inflation

- ✓ Temporarily raise or lower the # of tickets (in a cooperative env.)

9.3 Implementation

Benefit of Lottery scheduling

- ✓ Simplicity
  All it needs are 1) random(), 2) counter and 3) ticket at each job
- ✓ Example
  Three job (see figure)
  Assume that we pick the number 300 ? sched


```
1   // counter: used to track if we've found the winner yet
2   int counter = 0;
3
4   // winner: use some call to a random number generator to
5   //         get a value, between 0 and the total # of tickets
6   int winner = getrandom(0, totaltickets);
7
8   // current: use this to walk through the list of jobs
9   node_t *current = head;
10
11  // loop until the sum of ticket values is > the winner
12  while (current) {
13      counter = counter + current->tickets;
14      if (counter > winner)
15          break; // found the winner
16      current = current->next;
17  }
18  // 'current' is the winner: schedule it...
```

Figure 9.1: **Lottery Scheduling Decision Code**

9.4 An Example & 9.5 How to Assign Tickets?

Unfairness analysis

- ✓ Assumption: two jobs, same ticket, same run time (e.g. 10ms * N)
- ✓ U = C1/C2

  C1: Completion time of the earlier finished job
  C2: Completion time of the later finished job
  Implication (assume that N = 1)
  - C1=10, C2=20 ? U = 0.5 (worst fairness)
  - C1=20, C2=20 ? U = 1 (best fairness, ideal)
  - Long running ? Fig. 9.2

How to assign tickets?

- ✓ Money ? Cloud computing
- ✓ Priority ? Soft RT system
- ✓ …

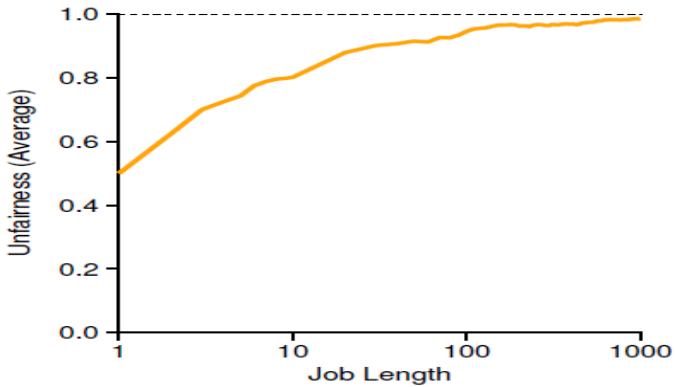


Figure 9.2: **Lottery Fairness Study**

9.6 Why Not Deterministic?

Lottery scheduling

- ✓ Not deterministic (rely on random number generator ? see 29 page)

Stride scheduling

- ✓ A deterministic fair-share scheduler

  Key concept: Stride ? Inverse in proportion to the # of tickets

  How to Schedule
  - Schedule a job who has the smallest pass value
  - Increment the pass value by its stride

- ✓ Example

  Three jobs: A, B, C, Tickets: 100, 50, 250

  Stride: 100, 200 and 40 (divide 10000 by ticket)

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
200	200	200	...

Figure 9.3: Stride Scheduling: A Trace

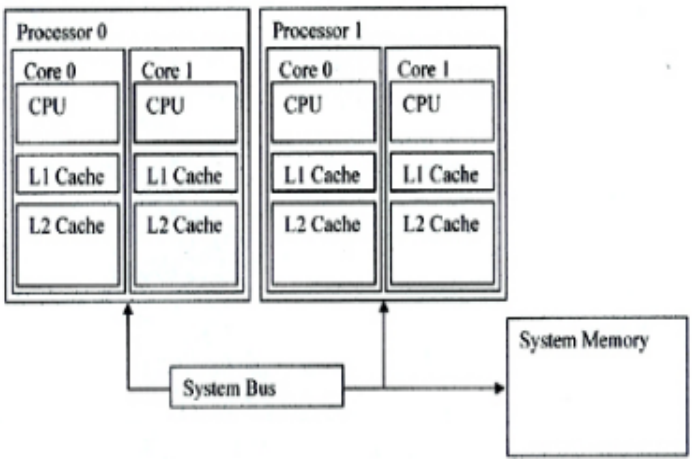
Chap. 10 Multiprocessor Scheduling (Advanced)

Multiprocessor and Multicore

- ✓ Multiprocessor: a system with multiple processors
- ✓ Multicore: a chip (socket, processor) with multiple cores
- ✓ Modern computer equips with multiple processors with multicore (with hyperthread) ? Manycore

For utilizing multicore effectively

- ✓ Typical programs: serial program (use only one CPU) ? make parallel program (e.g. using threads, Map/Reduce, …)
- ✓ Need a scheduler that can handle multiple CPUs ? load balancing



ASIDE: ADVANCED CHAPTERS

Advanced chapters require material from a broad swath of the book to truly understand, while logically fitting into a section that is earlier than said set of prerequisite materials. For example, this chapter on multiprocessor scheduling makes much more sense if you've first read the middle piece on concurrency; however, it logically fits into the part of the book on virtualization (generally) and CPU scheduling (specifically). Thus, it is recommended such chapters be covered out of order; in this case, after the second piece of the book.

10.1 Background: Multiprocessor Architecture

CPU cache (L1, L2, LLC)

- ✓ Small, fast memory that generally hold copies of popular data (based on temporal and spatial locality)
  - Temporal locality: when a data is accessed, it is likely to be accessed again in the near future (e.g. stack, for loop, …)
  - Spatial locality: when a data is accessed, it is likely to access data near as well (e.g. array, sequential execution, …)
- ✓ Benefit
  - Cache hit: make a program run fast by reducing access to the relatively slow main memory
  - Delayed write: modified data are kept in cache, not writing immediately into memory so that it possibly merges consecutive writes into a single memory access

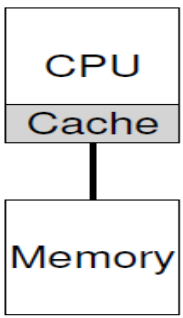


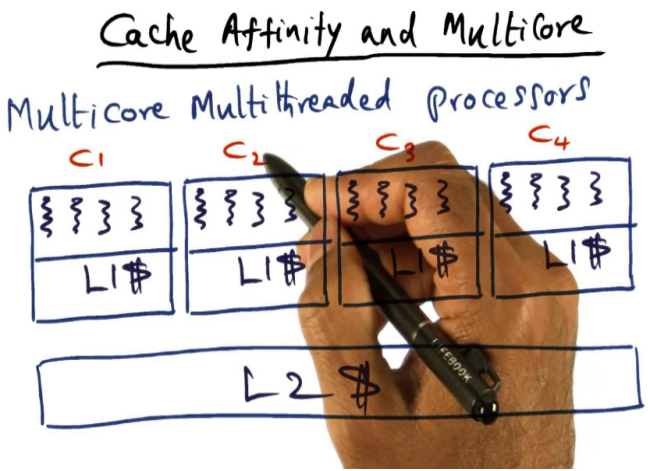
Figure 10.1: Single CPU With Cache

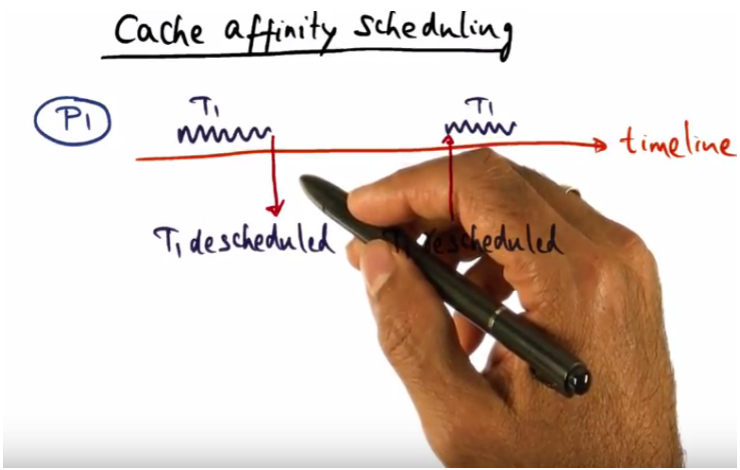
10.2 Synchronization & 10.3 Cache affinity

Issues on Multiprocessor

- Cache affinity

When a process runs, it is often advantageous to run it on the same CPU where the process ran previously
Since the CPU might build up a state in the cache (and TLB) for the process



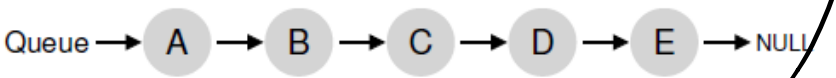


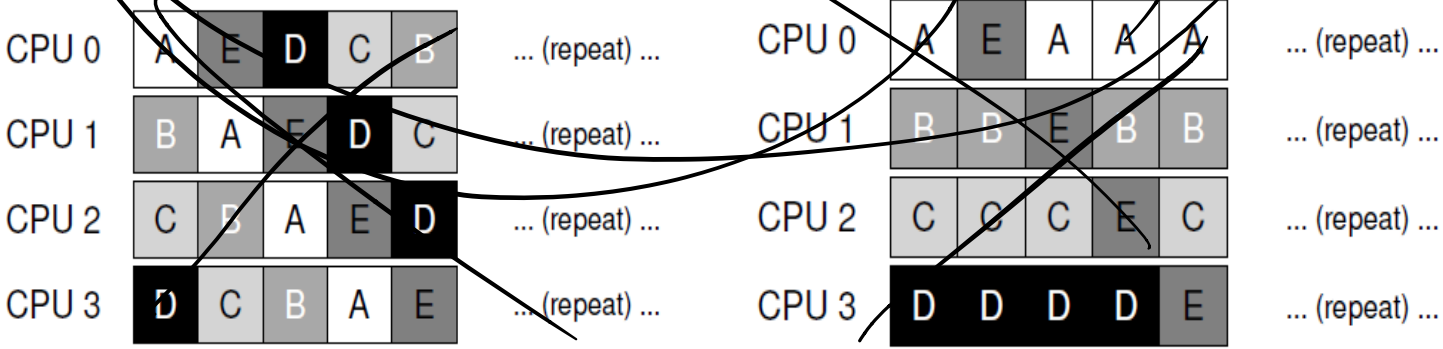
https://www.youtube.com/watch?v=fSUqT4WpPdM

10.4 Single-Queue Scheduling

SQMS (Single Queue Multiprocessor Scheduling)

- Use the framework for single processor scheduling
- Pros: simplicity
- Cons: cache affinity (5 jobs and 4 CPUs example, need to some complex mechanism to support cache affinity to obtain the below right figure), scalability (especially due to lock for shared queue)





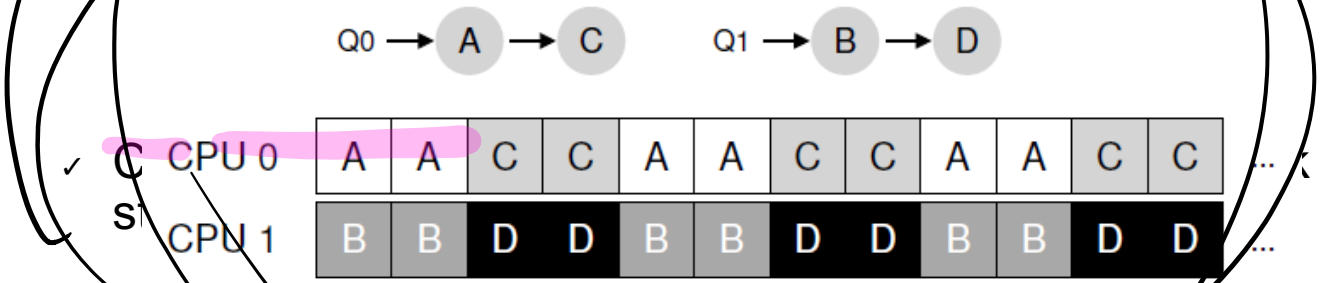
(without affinity consideration)

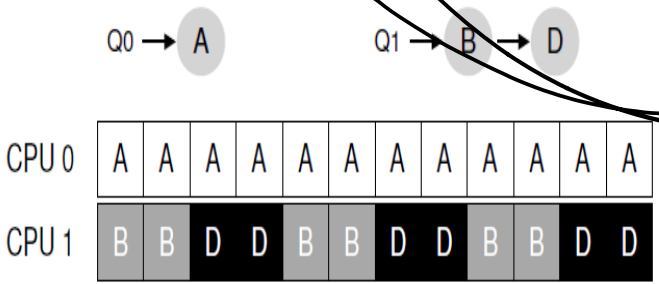
(with affinity consideration)

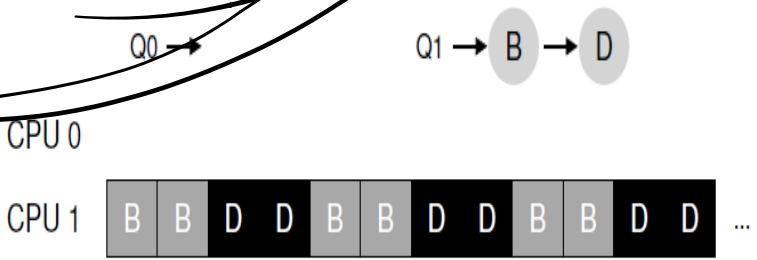
10.5 Multi-Queue Scheduling

MQMS (Multi-Queue Multiprocessor Scheduling)

- Multiple queues, Jobs assigned a queue, Each queue is associated with a CPU (or a set of CPUs)
- Pros: cache affinity, less lock contention



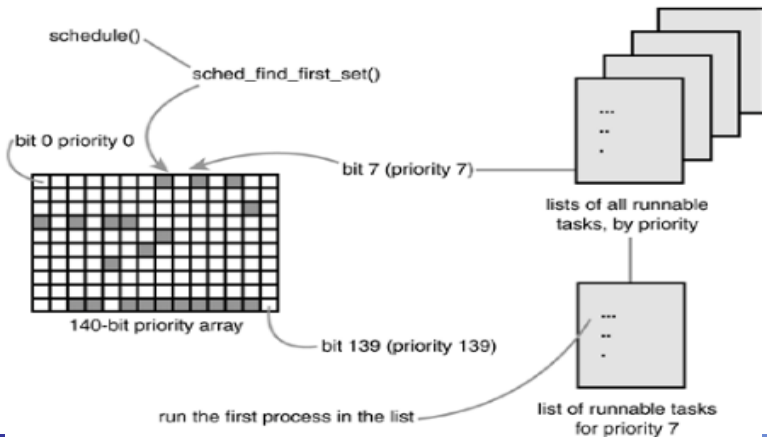


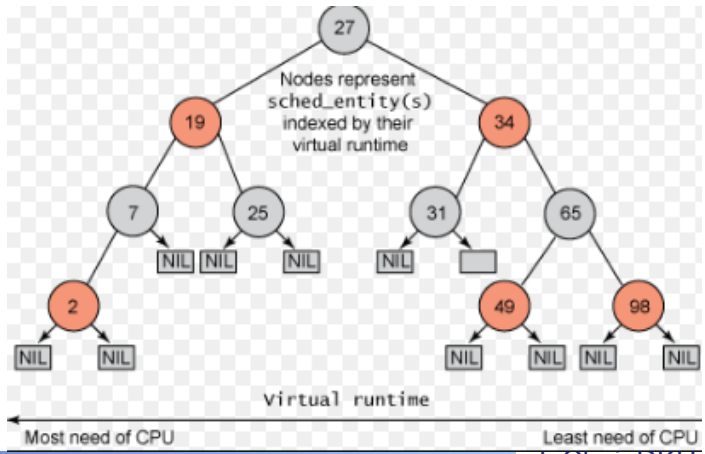


10.6 Linux Multiprocessor Schedulers (Optional)

Three different schedulers

- ✓ O(1) scheduler
  Multi-queue, similar to MLFQ (schedule higher priority, priority are changed dynamically)
- ✓ CFS (Complete Fair Share Scheduler)
  Multi-queue, similar to stride scheduling (deterministic proportional share scheduling)
- ✓ BF Scheduler
  Single-queue, proportional share with more complicate scheme





Chap 11. Summary Dialogue on CPU virtualization

What we have learned

- ✓ Mechanism: Context switch, Timer interrupt, Handler
- ✓ Policy: FCFS, SJF, RR, MLFQ, Lottery, Stride, Multiprocessor,

- How to build a team?
  - 1) 학생 2명이 자율적으로 한팀을 구성 (최대 2명, 3명 안됨!).
  - 2) 팀을 못 구성한 학생들은 나에게 이메일로 팀 구성 요청을 하세요. 그럼 제가 임의적으로 2명씩 연결하여 팀을 만들어 드리겠습니다.
  - 3) 만일 혼자서 Lab을 진행하고 싶은 학생은 나에게 혼자 진행하겠다고 이메일로 요청하세요.
  - 오늘 오후 (4월 8일) 6시까지 1, 2, 3 중에 해당되는 것을 나에게 이메일로 보내 주길 부탁합니다 (오늘은 이 메일로 출석 확인을 하겠습니다.)
  - ✓ Implementation: materialize as a real system
- Lab 1: Implement a scheduling simulator for FIFO, RR, MLFQ (e.g. page 24)
  - Requirement: 1) team (2 persons), 2) make a program, 3) at least two execution outputs (one workload same as 24 pages and different workloads).
  - How to submit? 1) report (3~5 pages for description and outputs) ? print out or email to professor, 2) Source code and report ? email to TA
  - Environment ? See Lab. 0 in the lecture site (and https://github.com/DKU-Embedded-Lab/2020_DKU_OS)
  - Due: until the same day of the next week (6PM. April 15th).
  - Bonus: Stride scheduler

Appendix 1: Real time scheduling

Task model: Ti (Ei, Di, Pi, Ai)

- ✓ Ei: execution time, Di: Deadline
- ✓ Pi: period if periodic task, Ai: arrival time

Scheduling algorithm

- ✓ EDF (Earliest Deadline First)
  Executes a job with the earliest deadline
- ✓ RM (Rate Monotonic)
  A task with a shorter period has a higher priority (Di= Pi in general)

Task	Execution Time	Period	Priority
T1	1	4	High
T2	2	6	Medium
T3	3	12	Low

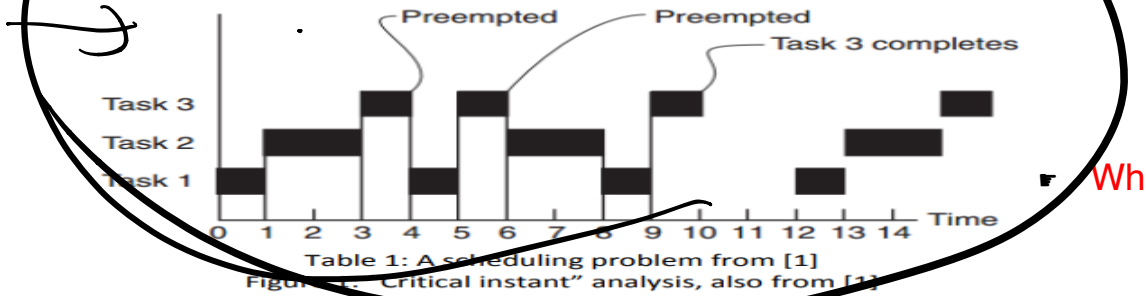


Table 1: A scheduling problem from [1]

Figure 1: "Critical instant" analysis, also from [1]

What if T1 (2,4)?

(Source: https://www.eecs.umich.edu/courses/eecs473/Labs/Lab3F17.pdf)

Appendix 2: 10.1 Multiprocessor Architecture

CPU cache is much complicated in Multiprocessor

- ✓ Cache coherence: maintain coherence among caches
  - A program running on CPU1 reads data from address A
  - CPU1 fetches the data and keep it its cache (assume its value is D)
  - The program modifies D into D'. CPU1 applies the delayed write
  - OS decides to schedule the program into CPU2 (due to load balancing)
  - The program re-read the value from address A.
  - The value is the old one(D), not the correct one (D') ? incoherent
- ✓ Bus snooping: one of mechanisms for supporting coherence
  - Monitoring cache, Invalidate or update if data is modified

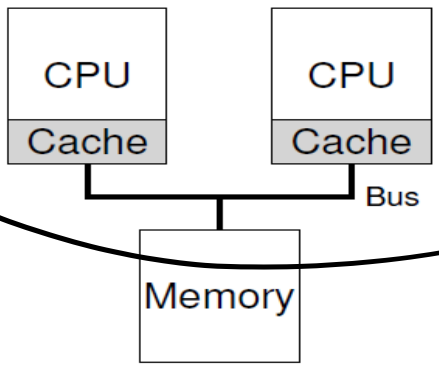


Figure 10.2: **Two CPUs With Caches Sharing Memory**

Appendix 2: 10.2 Don't Forget Synchronization

Another issues

- ✓ Mutual exclusion on shared data
  - Imagine if programs on two CPUs enter the List_Pop() routine at the same time
  - The first program executes line 9 while the second one executing line 8. What is the right content in the value (or head) variable?
  - May cause invalid pointer, double free, same value return, …
- ✓ Synchronization such as locking is required for correctness

```
1   typedef struct __Node_t {
2       int               value;
3       struct __Node_t *next;
4   } Node_t;
5
6   int List_Pop() {
7       Node_t *tmp = head;          // remember old head ...
8       int value   = head->value;   // ... and its value
9       head        = head->next;    // advance head to next pointer
10      free(tmp);                   // free old head
11      return value;                // return value at head
12  }
```

Figure 10.3: **Simple List Delete Code**