CS222
Operating Systems

Lecture 09
Linux CPU Scheduling: O(n) และ O(1)

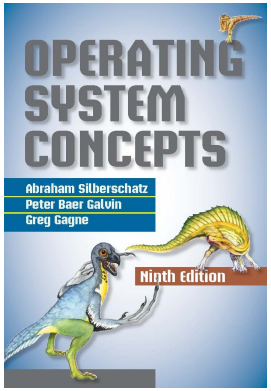
(Section 100001)

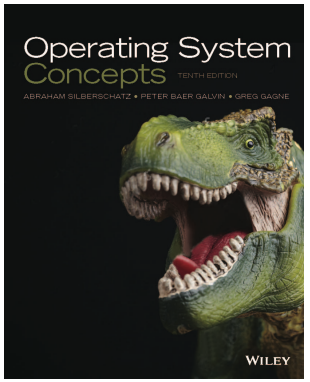
ผศ. ดร. กษิดิศ ชาญเชี่ยว

ckasidit@tu.ac.th

Textbook

- Avi Silberschatz, Peter B. Galvin and Greg Gagne; Operating System Concepts, 9th Edition; John Wiley & Sons, Inc; 2012; ISBN 978-1118063330





- Chapter 5

- Original Slides
- https://www.os-book.com/OS9/slide-dir/index.html
- https://www.os-book.com/OS10/slide-dir/index.html

Operating System Examples

- Linux scheduling
- ~~Windows scheduling~~
- ~~Solaris scheduling~~

Linux Scheduler

- เนื้อหาหลักนำมาจาก presentations จาก Indian Institute of Technology (IIT) ที่ Mudras
  https://www.youtube.com/watch?v=scfDOof9pww&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=21
- และจาก Indian Institute of Technology ที่ New Delhi
  https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

https://www.youtube.com/watch?v=scfDOof9pww&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=21
https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

Linux Define ชนิดของ Process

- Real Time Processes
  - การประมวลผลแต่ละรอบต้องเสร็จตาม Deadline
  - CPU ประมวลผลตาม Priority (ไม่ควรถูกแทรกโดย Process Priority ต่ำ)
- Normal Processes
  - Interactive
    - ใช้เวลาในการติดต่อกับผู้ใช้มาก มีการรอ I/O เช่น keyboard และ Mouse บ่อย
    - เมื่อมีป้อน input OS ควรปลุก Process ที่รอ I/O อยู่อย่างรวดเร็ว (มี delay ได้ระหว่าง 50 ถึง 150 ms เท่านั้น)
  - Batch
    - ไม่มีการปฏิสัมพันธ์ (interaction) กับผู้ใช้ในขณะที่ทำงาน มักจะรันเป็น **Background processes**

Real Time Process

- Real Time Processes
  - การประมวลผลแต่ละรอบต้องเสร็จตาม Deadline
  - CPU ประมวลผลตาม Priority (ไม่ควรถูกแทรกโดย Process Priority ต่ำ)
  - ถ้า Process เป็น Real Time แล้วก็เป็น Real Time ตลอด

Normal Process

- Normal Processes ซึ่งในระบบ Linux อาจเป็น Process หรือ Thread ก็ได้ บางที Linux จะเรียกมันว่า Task
  - Interactive
    - ใช้เวลาในการติดต่อกับผู้ใช้มาก มีการ**รอ** I/O เช่น keyboard และ Mouse บ่อย
    - เมื่อมีป้อน input OS ควรปลุก Process ที่รอ I/O อยู่อย่างรวดเร็ว (มี delay ได้ระหว่าง 50 ถึง 150 ms เท่านั้น)
  - Batch
    - ไม่มีการปฏิสัมพันธ์ (interaction) กับผู้ใช้ในขณะที่ทำงาน มักจะรันเป็น **Background processes**
  - Normal Process เปลี่ยนชนิดได้ เช่นเป็น interactive สักระยะแล้วเปลี่ยนเป็น batch
  - ระบบ Linux ใช้ heuristic เพื่อตัดสินว่า Process เป็น interactive หรือ batch โดยอัตโนมัติ ไม่ต้องให้ผู้ใช้กำหนด

Linux Scheduling History

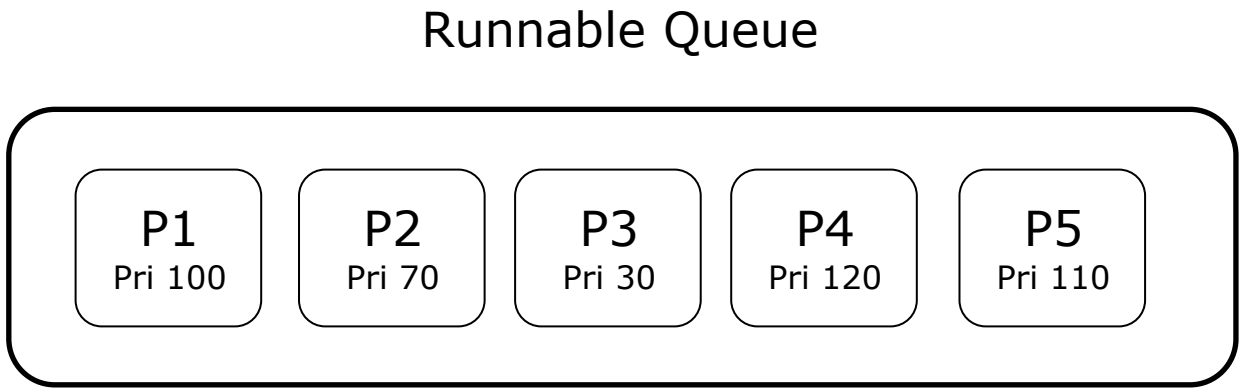
- O(n) Scheduler
  - Linux 2.4 to 2.6
- O(1) Scheduler
  - Linux 2.6 to 2.6.22
- Completely Fair Scheduler (CFS)
  - Linux 2.6.23 till Now

O(n) Scheduler

- เมื่อมีการ dispatch process
  - Scheduler จะ scan list ของ runnable process
  - เปรียบเทียบค่า priority ของแต่ละ Process
  - เลือก process ที่มี Priority สูงสุดจาก list
- ใช้เวลาหา Process โดยเข้าไปประเมิน Process ใน list N ครั้ง ถ้ามี N processes จึงเรียกว่า **O(N)** ซึ่งถ้ามีจำนวน N มากจะใช้เวลานาน (ไม่ Scalable)
  - ค้นพบปัญหานี้เมื่อมีการสร้าง java process เป็นจำนวนมากในระบบ
  - Big O เป็นสัญลักษณ์ของ Time Complexity
  - https://en.wikipedia.org/wiki/Big_O_notation
- ในกรณี SMP Multiprocessors ใช้ global run queue ให้ ซึ่ง scheduler สำหรับแต่ละ CPU core ต้องมาเลือก Process จาก runnable queue เดียวกัน
  - ไม่ Scalable (เมื่อจำนวน task มากขึ้นความเร็วลดลง)

https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

O(n) Scheduler



- N = 5
- ต้อง Scan Runnable Queue และพิจารณาทุก Item เพื่อหาอันที่มีค่า Priority น้อยที่สุด (หมายถึง Priority สูงสุด) ต้องพิจารณา 5 ครั้ง
- ถ้าเป็นระบบ SMP Multi-Processor ถ้าหลาย Processors ต้องดึง P จาก global queue นี้พร้อมกัน จะต้องมีการประสานกันเพื่อ ดึงข้อมูลออกมาจาก Queue ได้ทีละคน (ป้องกันการแข่งกัน race condition)

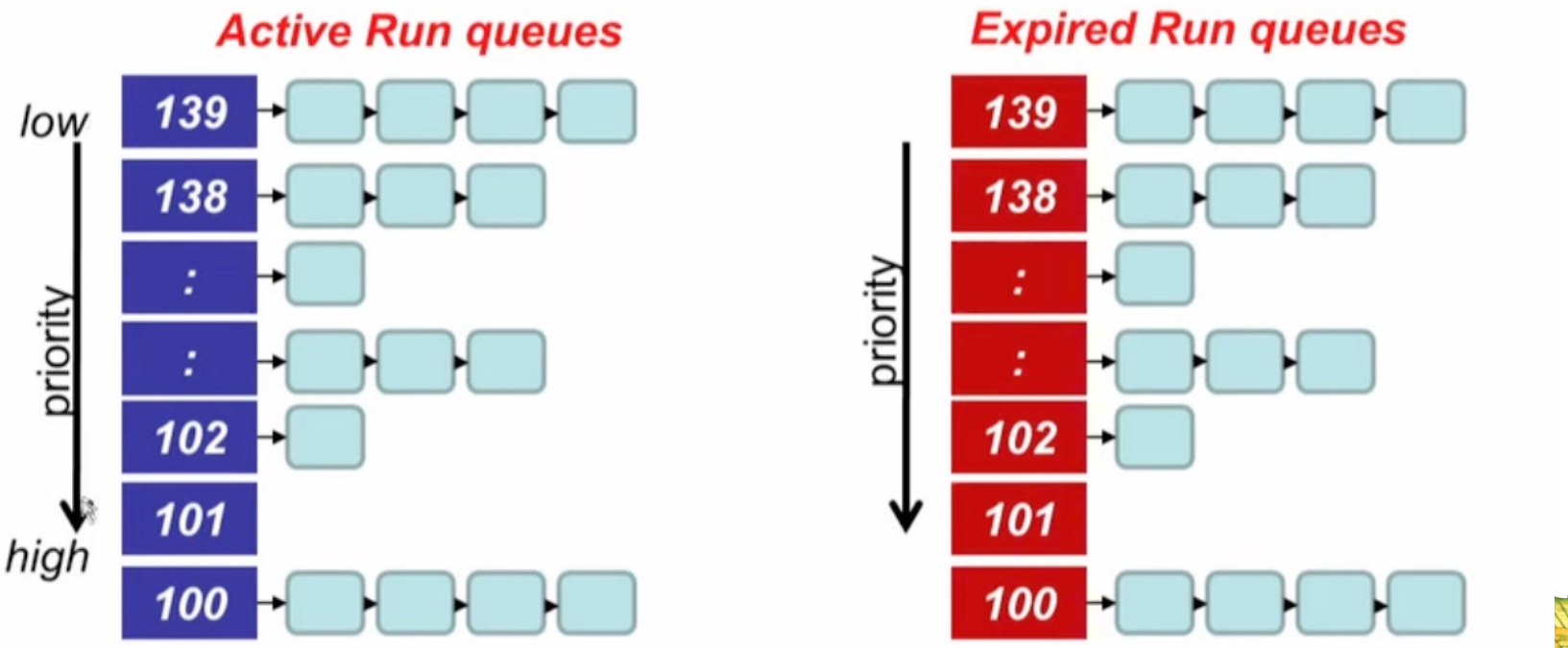
O(1) Scheduler

- **ใช้เวลาคงที่ (constant time)** ในการเลือก Process จาก runnable queue ไม่ว่าขนาดของจำนวนของ Process หรือ N ใน queue จะมากเท่าใด
  - Scalable
- แบ่ง Process เป็น 2 ชนิดได้แก่
  - Real TIme Processes
    - Priority จาก 0 ถึง 99
  - Normal Processes
    - Interactive
    - Batch
    - มีค่า Priority จาก 100 ถึง 139 โดยที่ 100 คือ Highest Priorrity และ 139 คือค่า Lowest Priority

https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

O(1) Scheduler สำหรับ Normal Processes

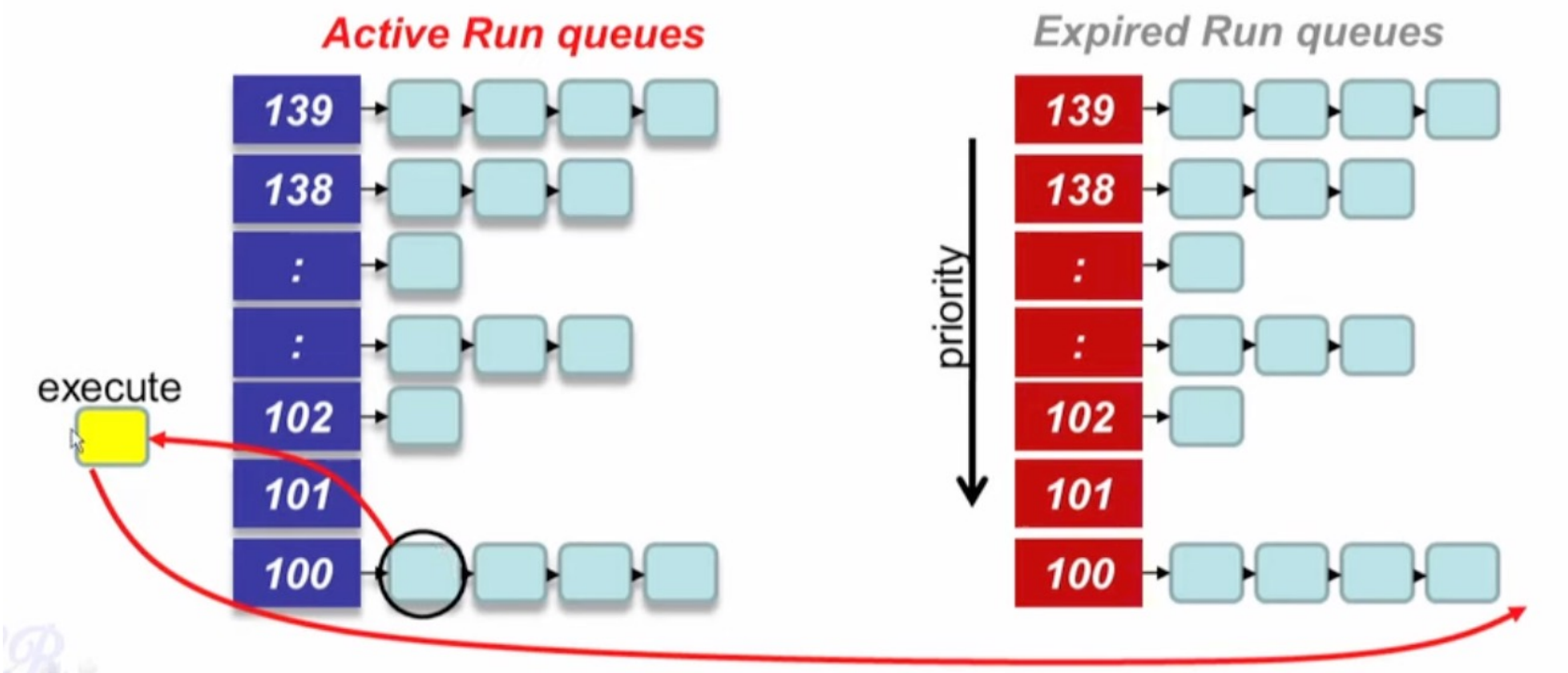
- Scheduler ชนิดนี้ใช้ Ready Queue 2 Queues เรียกว่า Active run queue และ Expired run queue ทั้งคู่เป็นแบบ Multilevel Queue
  - แต่ละ Queue มี 40 propriety classes
  - Priority 100 คือสูงสุด และ 139 คือต่ำสุด



https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

Scheduling Policy

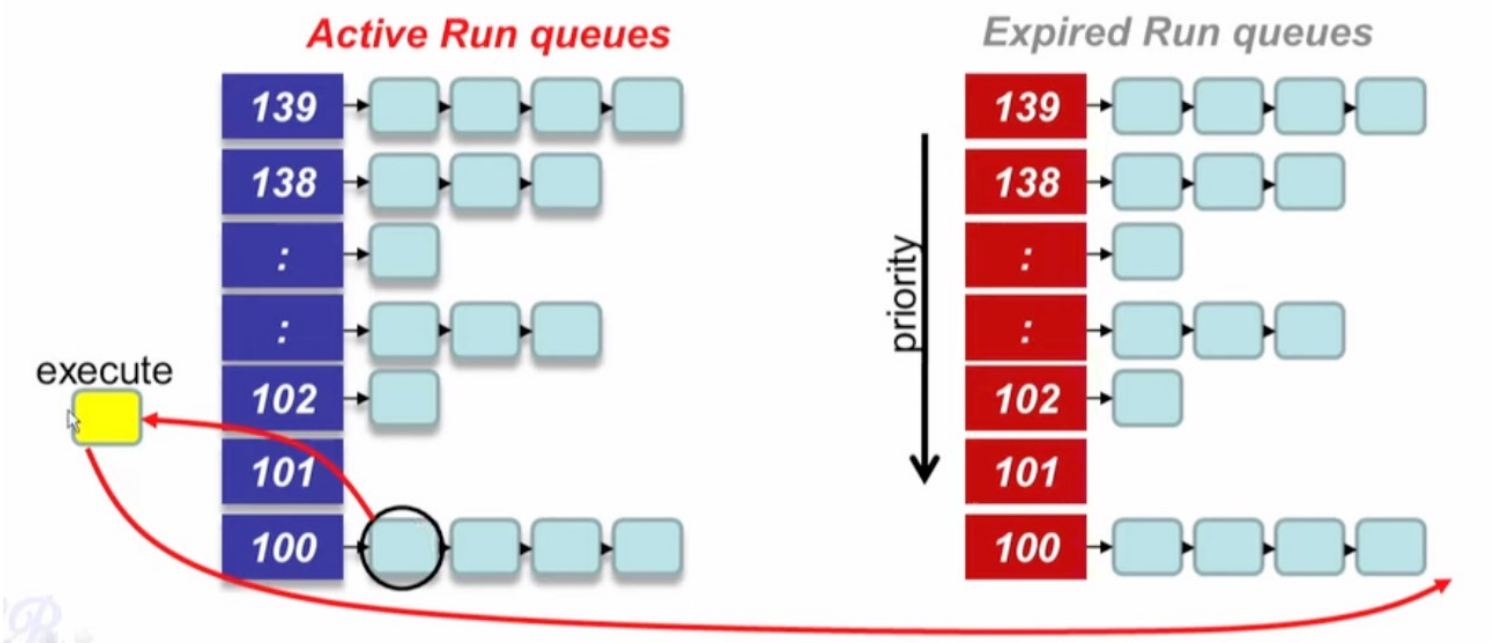
- เลือก Process (หรือ Task) ที่อยู่ต้น link list ของคิวใน active run queue ที่มี Priority level สูงสุด (หรือที่มีค่า priority น้อยที่สุด) จากแอคทีฟรันคิว
- เมื่อเลือก Task จนหมดลิงค์ลิสต์จากชั้นไพออริตี้หนึ่งแล้ว ก็จะเลือกจากลิงค์ลิสต์ในชั้นไพออริตี้ชั้นต่ำลงอันถัดไป



https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

Scheduling Policy

- เมื่อ process ประมวลผลหมด **time slice** จะถูกย้ายไป expire queue
- ถ้ายังไม่ไหมด time slice แต่ต้องรอ I/O มันจะถูกย้ายไป wait queue แต่ก็จะถูกย้ายกลับมาใน active queue (เมื่อรอ I/O เสร็จ) และถูกส่งให้ CPU ไปเรื่อยจนกว่า time slice หมด

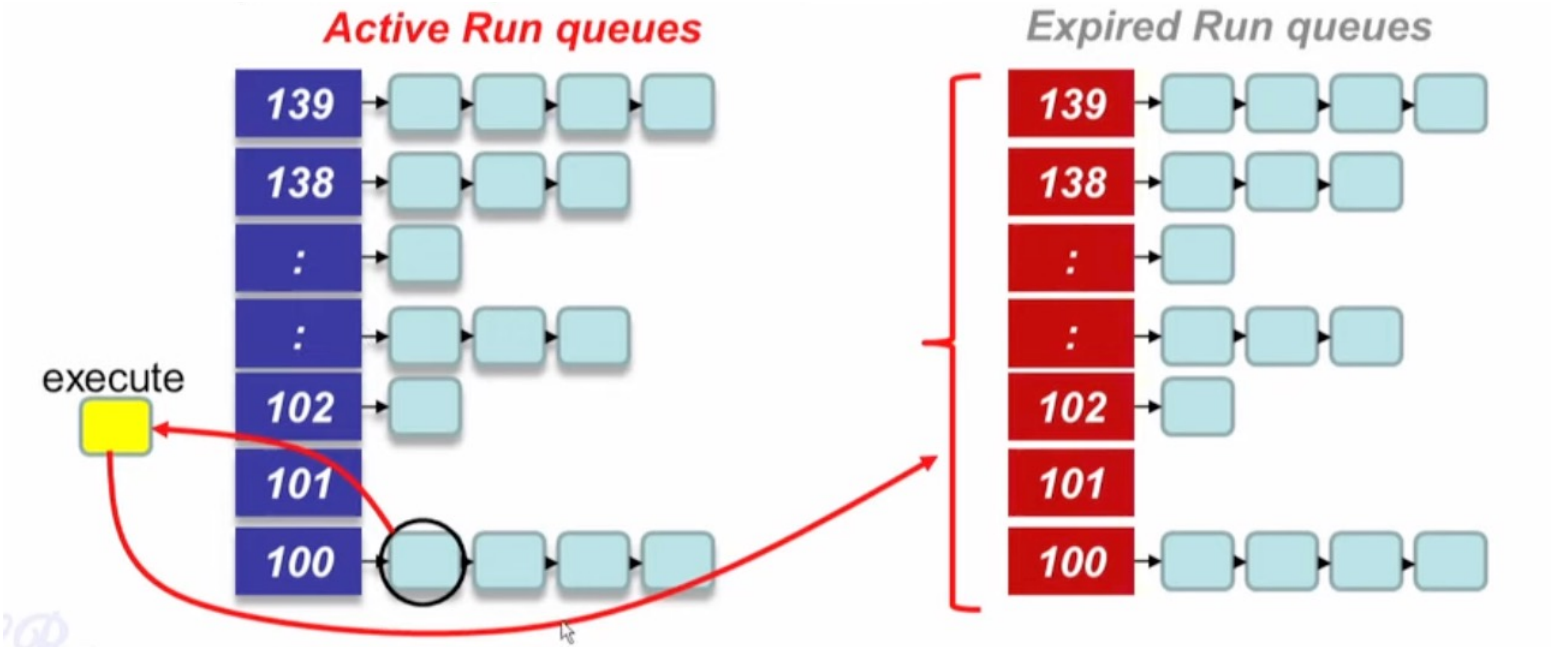


https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

https://en.wikipedia.org/wiki/O(1)_scheduler

Scheduling Policy

- เมื่อถูกนำไปใส่ใน expire queue ค่า priority ของ process นั้นจะถูกคำนวณใหม่และใส่ใน priority queue ที่หมาะสม
- จะกล่าวถึง สูตรที่ใช้เปลี่ยนค่า priority (เรียกว่า dynamic priority) ใน slide ถัดไป

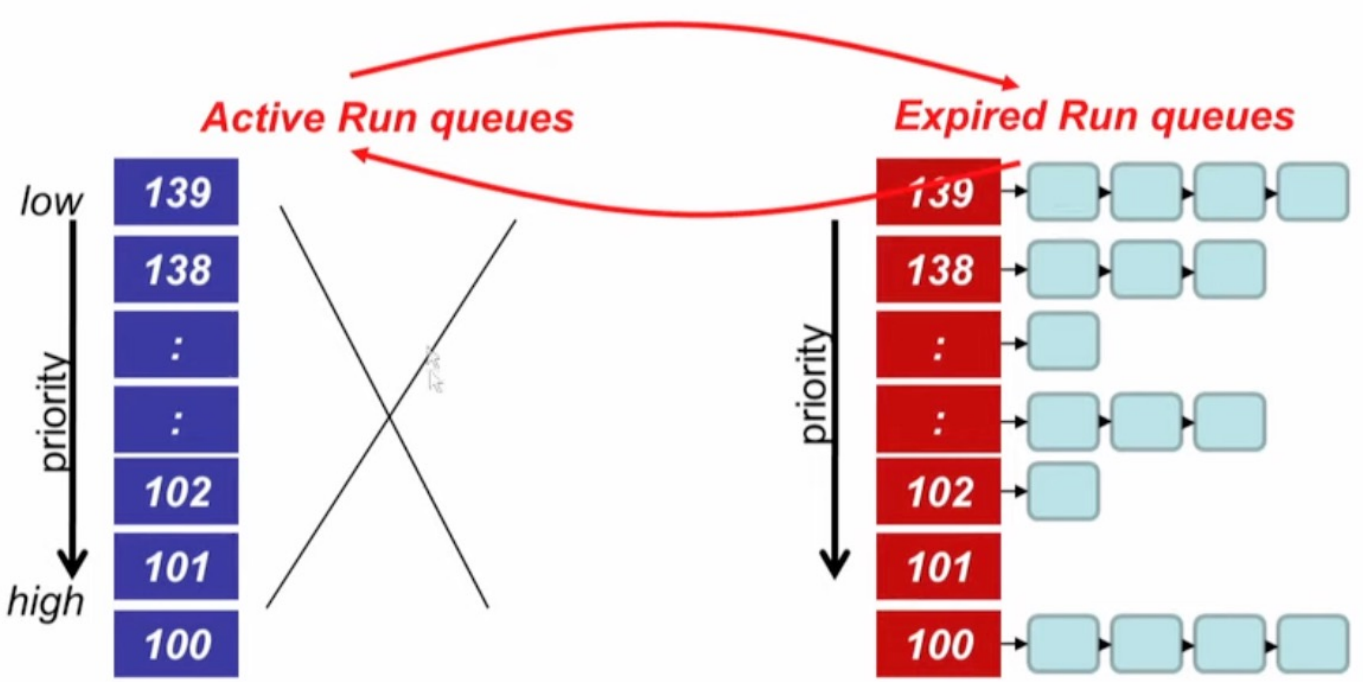


https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

https://en.wikipedia.org/wiki/O(1)_scheduler

Scheduling Policy

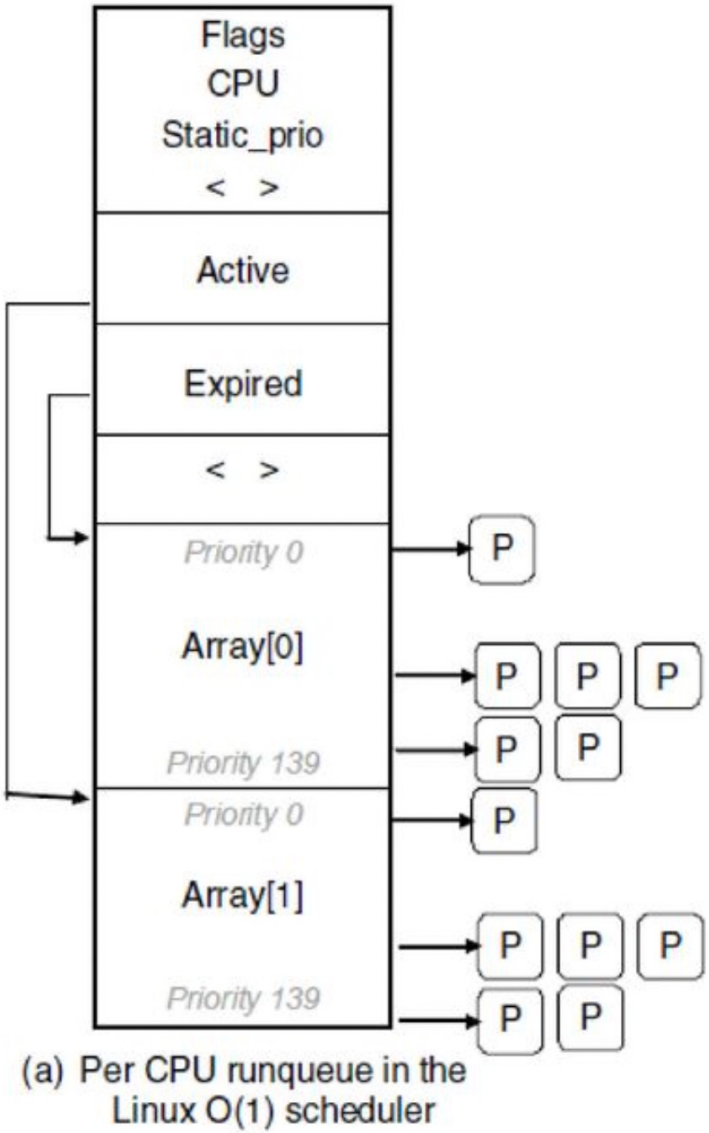
- เมื่อ active queue ว่างแล้ว จะ swap active และ expire queue



https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

Q implementation

- Linux ใช้ pointer สอง Pointer ชี้ไปยังพื้นที่ในหน่วยความจำแบบ Array ของ Pointer ที่เก็บข้อมูลของ รันคิวทั้งสอง
- เวลาที่ต้อง Swap (สลับ) คิว ก็แค่สลับ “Active” และ ”Expired” pointers ในภาพ



(a) Per CPU runqueue in the Linux O(1) scheduler

https://slideplayer.com/slide/12965396/

การเลือก task ด้วยเวลาคงที่ใน O(1) scheduler

- มีการทำงาน 2 ขั้นตอน ในการเลือก process เข้าใช้ CPU
  - Step 1: หา active run queue **ที่มีค่า rank ต่ำที่สุด** ที่มี process อยู่
  - Step 2: เลือก Task จาก queue นั้น
- **พิจารณา Step 1:**
- ใน Step 1 การหา queue ที่มี rank ต่ำสุดต้องเริ่มไล่หาจาก 100 ไปเรื่อย ๆ ถ้าหาไม่เจอที่ 100 ก็เพิ่มไปหาที่ 101 ไล่ไปจนถึง 139
- **ไม่ใช่เวลาคงที่**
  - ถ้า task แรกที่เจออยู่ที่ run queue priority 100 ก็จะได้เร็ว
  - แต่ถ้า task แรกที่ไล่หาแล้วเจออยู่ที่ run queue priority 109 ก็ต้องจะใช้เวลา **หานานกว่า**กรณี queue 100 ถึง 9 step

https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

การเลือก task ด้วยเวลาคงที่ใน O(1) scheduler

- มีการทำงาน 2 ขั้นตอน ในการเลือก process เข้าใช้ CPU
  - Step 1: หา active run queue **ที่มีค่า rank ต่ำที่สุด** ที่มี process อยู่
  - Step 2: เลือก Task จาก queue นั้น
- **พิจารณา Step 1:**
- วิธีแก้ปัญหา CPU ต้องมีคำสั่ง ISA "find-first-bit-set"
  - e.g. Intel ใช้คำสั่ง `bsfl`
- OS จะต้อง maintain bit vector ที่ 1 bit เป็นตัวแทนของ 1 run queue priority เมื่อมีการเปลี่ยนแปลงใน run queue OS ต้อง update bit vector ด้วย

```
int queuebits ="0b00010"; // ตัวแทน run queue priority
int bit_position; // ผลลัพธ์
__asm__("bsfl  bit_position, queuebits")
// จะได้ผลลัพธ์เป็น 1
set_priority_level = bit_position + 100;// จะได้เป็น run queue 101
```

- จะรู้ภายในคำสั่งเดียวว่า run queue ต่ำที่สุดที่มี task คืออะไร

https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

การเลือก task ด้วยเวลาคงที่ใน O(1) scheduler

- มีการทำงาน 2 ขั้นตอน ในการเลือก process เข้าใช้ CPU
  - Step 1: หา active run queue **ที่มีค่า rank ต่ำที่สุด** ที่มี process อยู่
  - Step 2: เลือก Task จาก queue นั้น
- พิจารณา Step 2:
- ใน Step 2 ใช้เวลาคงที่ เพราะเลือก Task จาก Head ของ Link List
- สรุป
  - Step 1 ใช้เวลาคงที่เพราะใช้ hardware CPU ISA support
  - Step 2 คงที่เพราะเอาจากต้น list
  - รวมคือใช้เวลาคงที่ในการหา process จาก queue ทุกครั้ง

https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

Static Priority และ Nice levels

- Static Priorities คือการกำหนดค่า Priority ตั้งต้น (initial) ของ Process โดย**ผู้ใช้** แต่ในระหว่างที่ Process ประมวลผลค่า Priority ของมันจะเปลี่ยนตามหลักการ Dynamic Priority
- Real Time Processes: ค่า Priority 0 – 99
- Normal Processes:
  - 100 คือ Priority สูงสุด
  - 139 คือ Priority ต่ำสุด
  - 120 คือค่า base priority เป็นค่า Default
- nice คือคำสั่งให้ผู้กำหนดค่า default static priority ของ Process

  $ nice -n N ./a.out
- N คือค่า ตั้งแต่ +19 ถึง -20
  - ค่า priority สูงสุด ค่า nice = -20 (จะขอรันก่อน process อื่น)
  - ค่า priority ต่ำสุด ค่า nice = 19 (ให้ process อื่นรันก่อน)
  - ค่า priority กลาง ค่า nice = 0 (base priority)

heuristic คืออะไร

- วิธีการ Heuristic technique คือวิธีการในการแก้ปัญหาด้วยความคิดหรือหลักการที่สร้างขึ้นด้วยตนเอง ที่นำไปใช้ได้ง่ายในทางปฏิบัติซึ่งหลักการนั้นไม่จำเป็นต้อง
  - เป็นวิธีการที่ดีที่สุด
  - เป็นวิธีการที่สมบูรณ์ที่สุด
  - เป็นวิธีการที่สมเหตุสมผล

  แต่เป็นการคาดคะเนหรือการประมาณที่ ดีเพียงพอที่สามารถนำไปใช้แก้ปัญหาเฉพาะหน้าได้

- อาจได้มากจาก การลองผิดลองถูก (trials and errors) กฎจากหลักการอย่างง่าย (rules of thumb) การคาดเดาด้วยหลักการแบบง่าย (educated guess)
- https://en.wikipedia.org/wiki/Heuristic

Dynamic Priority

- **Dynamic Priorities** คือการกำหนดค่า Priority ที่ค่า Priority จะเปลี่ยนไปเรื่อย ๆ ในระหว่างที่ Process ประมวลผล (มี Static Priority เป็นค่าตั้งต้น)
- Normal Process สามารถเปลี่ยนจาก Interactive Process เป็น Batch Process หรือกลับกันได้
- Linux ให้ Interactive Process มี Dynamic Priority สูงกว่า Batch Process
- **Linux ใช้ Heuristic** ในการพิจารณาค่า Dynamic Priority ของแต่ละ Process
  - ถ้า Process Sleep มาก (เช่นรอ I/O มาก) เป็นอาการของ Interactive process ควรได้รับการตอบสนองอย่างรวดเร็ว Linux จะกำหนดให้ Process นั้นมี Dynamic Priority สูง (ค่า run queue level ต่ำ)
  - ถ้า Process Sleep น้อย (เช่นรอ I/O มาก) เป็นอาการของ Batch process Linux จะให้มี Dynamic Priority ต่ำ (ค่า run queue level สูง)

Heuristic สำหรับกำหนด Dynamic Priority

- Linux จะกำหนดค่า Dynamic Priority **หลังจากที่ Process จบการใช้งาน CPU**
- Linux จะนำ Task นั้นไปใส่ใน Expired run queue ที่มีค่า priority ตรงกับค่า Dynamic run queue ใหม่ที่คำนวณได้

Heuristic สำหรับกำหนด Dynamic Priority

- Linux จะใช้ Heuristic ต่อไปนี้ในการกำหนดค่า Dynamic Priority

```
K = MIN( static_priority – bonus + 5, 139)    // สูงสุดไม่มากกว่า 139
dynamic_priority = MAX(100, K)                // ต่ำสุดไม่น้อยกว่า 100
```

- ค่าตัวแปร bonus จะมีค่าตั้งแต่ 0 ถึง 10
- ถ้า bonus มีค่า < 5 หมายความว่า Process มี Interaction น้อย และเป็น CPU bound process ควรถูกลด dynamic priority (มีค่า run queue level สูงขึ้น toward 139 ขอให้สังเกตว่าค่า **– bonus + 5** จะเป็นค่า +)
- ถ้า bonus มีค่า > 5 หมายความว่า Process มี Interaction มาก และเป็น I/O bound process ควรเพิ่ม dynamic priority (มีค่า run queue level ลดลง toward 100)

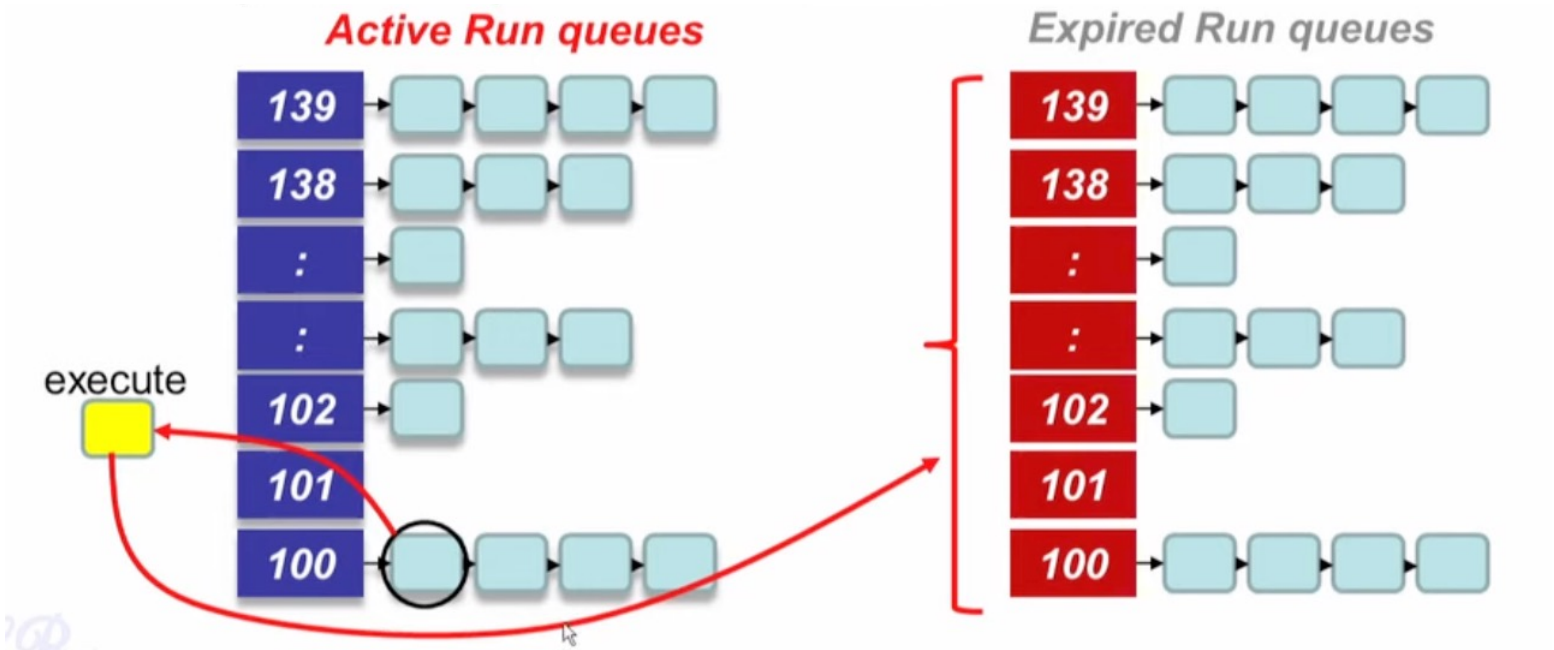
Heuristic สำหรับกำหนดค่า Bonus

- I/O bound process หลับมากกว่าควรได้ priority สูงขึ้น (จาก static pri ของมัน)
- CPU bound process หลับน้อยกว่าควรได้ priority ต่ำลง (จาก static pri)
- Linux กำหนดค่า bonus โดยใช้ค่า Average Sleep Time ของ Process ตามกฎแบบ Heuristic ต่อไปนี้

average sleep time "s" (miliseconds)	bonus
$0 \leq s < 100$	0
$100 \leq s < 200$	1
$200 \leq s < 300$	2
$300 \leq s < 400$	3
$400 \leq s < 500$	4
$500 \leq s < 600$	5
$600 \leq s < 700$	6
$700 \leq s < 800$	7
$800 \leq s < 900$	8
$900 \leq s < 1000$	9
$1\ sec$	10

กำหนด Run Q ตาม Dynamic Priority

- O(1) scheduler เลือก Task จาก Active run Queue แล้วส่งให้ CPU
- หลังจาก CPU รัน Task นั้นแล้ว Linux จะนำค่า Bonus มาหาค่า Dynamic priority ที่เหมาะสมตาม heuristic แล้วใส่ Task ใน Expire run queue ตามค่า Dynamic priority ที่คำนวณได้



การกำหนดค่า Timeslice

- Time Slice หรือ Time Quantum คือระยะเวลาที่ Scheduler กำหนดให้ Task (Process หรือ Thread) ประมวลผลบนซีพียู (Task จะใช้ CPU น้อยกว่าหรือเท่ากับเวลาที่กำหนดใน Time Slice ก็ได้ แต่ไม่มีวันเกินระยะเวลา Time Slice)
- Linux's O(1) Scheduler จะคำนวณค่า Time Slice ของ Task ตาม Priority ของ Normal Task (หรือ Normal Process/Thread) โดยมีหลักการคือ
  - Task ที่มี **Priority สูง**เช่น Interactive Task ควรได้ Time Slice มาก
  - Task ที่มี **Priority ต่ำ**เช่น Batch Task ควรได้ Time Slice น้อย

การกำหนดค่า Timeslice

- ใช้ Heuristic ต่อไปนี้
  - Linux ใช้ priority = static priority ในการคำนวณครั้งแรก
  - หลังจากนั้นมันจะใช้ priority = dynamic priority

```
if (priority_n < 120)
     time_slice = (140 – priority_n) * 20 millisec
Else
     time_slice = (140 – priority_n) * 5 millisec
```

priority	priorty_n	niceness	quantum
สูงสุด	100	-20	800 ms
สูง	110	-10	600 ms
ปกติ	120	0	100 ms
ต่ำ	130	10	50 ms
ต่ำสุด	139	19	5 ms

Linux Scheduling Through Version 2.5

- Prior to kernel version 2.5, ran variation of standard UNIX scheduling algorithm
- Version 2.5 moved to constant order *O*(1) scheduling time
  - Preemptive, priority based
  - Two priority ranges: time-sharing and real-time
  - **Real-time** range from 0 to 99 and **nice** value from 100 to 140
  - Map into global priority with numerically lower values indicating higher priority
  - Higher priority gets larger q (time quantum)
  - Task run-able as long as time left in time slice (**active**)
  - If no time left (**expired**), not run-able until all other tasks use their slices
  - All run-able tasks tracked in per-CPU **runqueue** data structure
    - Two priority arrays (active, expired)
    - Tasks indexed by priority
    - When no more active, arrays are exchanged
  - Worked well, but poor response times for interactive processes

ข้อจำกัดของ O(1) Scheduler

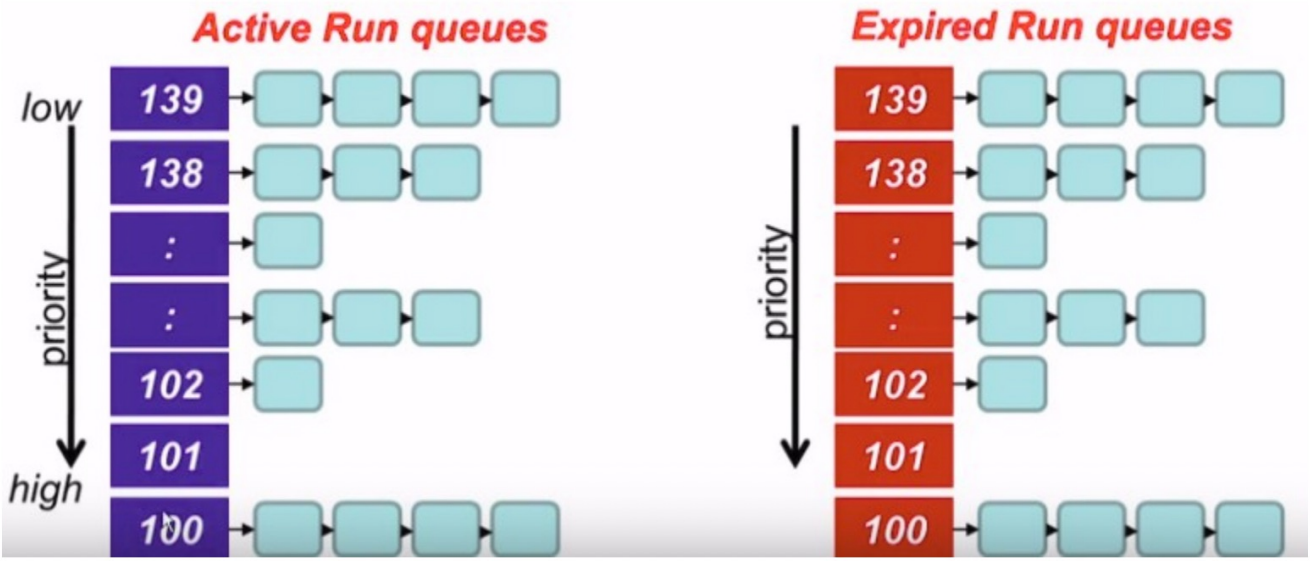
- ใช้ Heuristic มาก มี Heuristic ที่ซับซ้อน ที่ใช้ตัดสิน Interactive และ Batch Tasks
- ไม่แฟร์เพราะ การเปลี่ยนแปลงของ Priority กับการเปลี่ยนแปลงค่า Time Slice ไม่มีมาตรฐาน ไม่ Uniform

priority	priorty_n	quantum
สูงสุด	100 (-10)	800 ms (200+)
สูง	110 (-10)	600 ms (500+)
ปกติ	120 (-10)	100 ms (50+)
ต่ำ	130 (-9)	50 ms (45+)
ต่ำสุด	139	5 ms

ชั้น	ไม่แฟร์	แฟร์
5	10	10
4	6	8
3	4	6
2	2	4
1	2	2

สรุป O(1) Scheduler

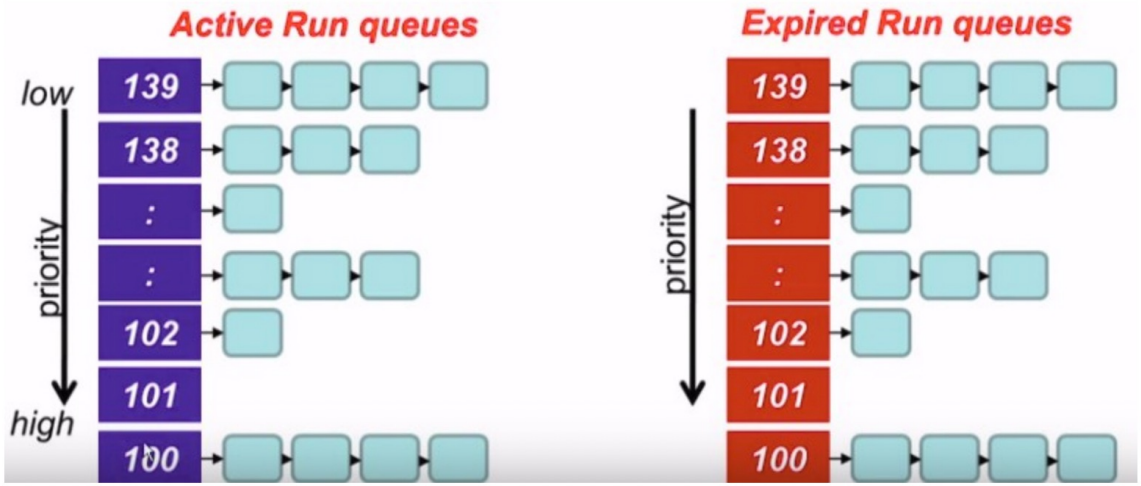
- ใช้หลักการ priority queue
- มี queue สองแบบคือ Active และ Expired run queue
- แต่ละ queue มีระดับ priority 40 อันดับ
- Priotity 100 = สูงสุด Priotity 139 = ต่ำสุด



https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

สรุป O(1) Scheduler

- เมื่อ process ประมวลผลหมด time slice จะถูกย้ายไป expire queue
- ถ้ายังไม่ไหมด time slice แต่ต้องรอ I/O ก็จะยังอยู่ใน active queue และถูกส่งให้ CPU ไปเรื่อยจน time slice หมด
- เมื่อถูกนำไปใส่ใน expire queue ค่า priority ของ process นั้นจะถูกคำนวณใหม่และใส่ใน priority queue ที่หมาะสม
- เมื่อ active queue ว่างแล้ว จะ swap active และ expire queue



https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

https://en.wikipedia.org/wiki/O(1)_scheduler

O(1) Scheduler: คำนวณค่า Prioiry

- ค่า Priority ใช้แยก I/O-bound และ CPU-bound process
- ใช้ค่าเฉลี่ยของการ sleep ของแต่ละ process ในการคำนวณ
- I/O-bound process จะ sleep มากกว่า จึงมี priority สูงกว่า
- CPU-bound process จะ sleep น้อย จึงมี priority ต่ำกว่า
- ปัญหา: ยากที่จะค่า time slice ที่เหมาะสม สัดส่วนค่า Time Slice ไม่แฟร์

https://www.cse.iitd.ac.in/~rijurekha/col788/scheduling1.pdf

สรุป

- หลักการทั่วไปของ Linux Scheduler
- O(n) Scheduler
- O(1) Schedule โครงสร้าง และกลไก (Mechanism)
- Heuristic ที่ใช้กำหนดค่าสำคัญใน O(1)
- ข้อจำกัดของ O(1)