CPU Scheduling

Week 3: Penjadwalan CPU — FIFO, SJF, RR, dan MLFQ

lectura.id/course/os

Program Studi Teknik Informatika
Institut Teknologi Sumatera

2026

OUTLINE

Peta materi pertemuan ini

Capaian Pembelajaran

Kemampuan yang ditargetkan setelah kuliah

1. Memahami mengapa CPU perlu dijadwalkan
2. Mengenal metrik penjadwalan: turnaround time dan response time
3. Menjelaskan algoritma FIFO, SJF, STCF, dan Round Robin
4. Memahami trade-off antara turnaround time dan response time
5. Menjelaskan cara kerja Multi-Level Feedback Queue (MLFQ)
6. Menganalisis masalah starvation, gaming, dan solusinya

Target Akhir

Setelah pertemuan ini, mahasiswa mampu membandingkan algoritma penjadwalan dan memahami mengapa MLFQ menjadi solusi **praktis** di OS modern.

Mengapa Perlu Penjadwalan?

Masalah mendasar di balik scheduler

Bayangkan sebuah kedai kopi dengan **satu barista** dan **banyak pelanggan**.

1. Siapa yang dilayani duluan?
2. Apakah pelanggan dengan pesanan kecil harus menunggu pelanggan dengan pesanan besar?
3. Bagaimana agar semua merasa dilayani dengan adil?

Inti Masalah

CPU itu seperti barista tunggal. **Scheduler** adalah aturan antrian yang memutuskan proses mana yang dijalankan berikutnya.

Kebijakan vs Mekanisme

Dua level keputusan penjadwalan

Mekanisme (HOW)

1. Context switch
2. Timer interrupt
3. Trap ke kernel

Kebijakan (WHAT)

1. Proses mana yang jalan duluan?
2. Berapa lama giliran tiap proses?
3. Kapan proses dihentikan paksa?

Hubungan Keduanya

Minggu lalu kita belajar **mekanisme** (context switch, mode kernel). Sekarang kita belajar **kebijakan** penjadwalan.

Asumsi Beban Kerja

Penyederhanaan agar mudah dipahami

Untuk memahami algoritma secara bertahap, kita mulai dengan 5 asumsi (yang nanti akan kita longgarkan satu per satu):

1. Setiap pekerjaan berjalan dalam durasi yang **sama**
2. Semua pekerjaan **tiba bersamaan**
3. Pekerjaan **tidak bisa dihentikan** di tengah (non-preemptive)
4. Semua pekerjaan **hanya pakai CPU** (tidak ada I/O)
5. Durasi setiap pekerjaan **sudah diketahui**

Kenapa Asumsi Ini Penting?

Dengan asumsi sederhana, kita bisa menganalisis algoritma secara **matematis**. Setelah paham, kita longgarkan asumsinya.

Metrik: Turnaround Time

Mengukur seberapa cepat pekerjaan selesai

Definisi Formal

$$T_{\text{turnaround}} = T_{\text{selesai}} - T_{\text{tiba}}$$

Analogi: Turnaround time pesanan kopi = waktu kopi jadi **dikurangi** waktu Anda pesan.

1. Nilai makin **kecil** = makin **bagus**
2. Jika semua tiba di waktu 0, maka turnaround = waktu selesai
3. Ini adalah metrik **kinerja** (bukan keadilan)

Metrik: Response Time

Mengukur seberapa cepat pekerjaan mendapat giliran pertama

Definisi Formal

$$T_{\text{response}} = T_{\text{pertama dijadwalkan}} - T_{\text{tiba}}$$

Analogi: Response time = waktu barista **pertama kali menyapa** Anda setelah Anda datang.

1. Penting untuk sistem **interaktif** (terminal, GUI, game)
2. Berbeda dari turnaround: response hanya soal “kapan mulai dilayani”
3. Turnaround time bisa bagus tapi response time buruk (atau sebaliknya!)

FIFO (First In, First Out)

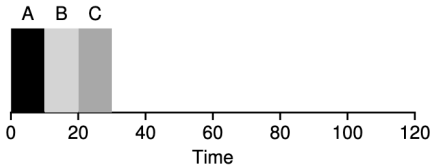
Algoritma paling sederhana: siapa datang duluan, dilayani duluan

Tiga pekerjaan: A, B, C — masing-masing 10 detik.

Rata-rata turnaround:

$$\frac{10 + 20 + 30}{3} = 20 \text{ detik}$$

Sederhana dan adil — **selama** semua pekerjaan sama panjangnya!



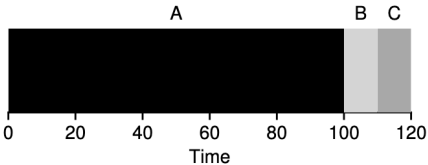
Masalah FIFO: Convoy Effect

Pekerjaan pendek terjebak di belakang pekerjaan besar

A = 100 detik, B = C = 10 detik.

Rata-rata turnaround:

$$\frac{100 + 110 + 120}{3} = 110 \text{ detik}$$



Seperti antri di kasir: satu orang dengan troli penuh membuat semua orang menunggu lama!

Convoy Effect

Pekerjaan pendek “terkonvoi” di belakang pekerjaan besar. Turnaround meledak!

Latihan: FIFO

Uji pemahaman Anda tentang FIFO dan convoy effect

Soal

Tiga proses tiba bersamaan di $t = 0$ dengan durasi: P1 = 8 detik, P2 = 4 detik, P3 = 2 detik. Urutan kedatangan adalah P1, P2, P3.

Dengan algoritma **FIFO**, hitung:

1. Turnaround time masing-masing proses
2. Rata-rata turnaround time
3. Apakah terjadi convoy effect? Jelaskan!

Petunjuk

Ingat: FIFO menjalankan proses sesuai urutan kedatangan, tanpa menyela.

Jawaban: FIFO

Pembahasan soal latihan FIFO

Urutan eksekusi: P1 (0–8), P2 (8–12), P3 (12–14)

Proses	Durasi	Selesai	Turnaround
P1	8	8	8
P2	4	12	12
P3	2	14	14

$$\text{Rata-rata turnaround} = \frac{8 + 12 + 14}{3} \approx 11.3 \text{ detik}$$

Convoy Effect: Ya!

P3 (hanya 2 detik) harus menunggu 12 detik sebelum jalan, karena terjebak di belakang P1 yang panjang. Jika urutan dibalik (P3, P2, P1), rata-rata turnaround hanya $\frac{2+6+14}{3} \approx 7.3$ detik.

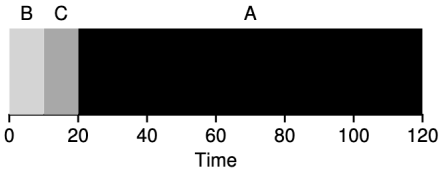
SJF (Shortest Job First)

Jalankan pekerjaan terpendek terlebih dahulu

Ide: B dan C (10 detik) dijalankan **sebelum** A (100 detik).

Rata-rata turnaround:

$$\frac{10 + 20 + 120}{3} = 50 \text{ detik}$$



Turun drastis dari 110 menjadi 50! Seperti kasir supermarket yang punya jalur “10 item atau kurang”.

Fakta Menarik

SJF **optimal** untuk turnaround time — selama semua pekerjaan tiba bersamaan.

Masalah SJF: Pekerjaan Tidak Selalu Tiba Bersamaan

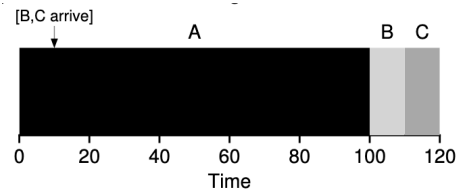
Apa yang terjadi jika pekerjaan pendek datang terlambat?

A tiba di $t = 0$ (100 detik). B dan C tiba di $t = 10$ (masing-masing 10 detik).

A sudah berjalan $\rightarrow$ B dan C harus menunggu A selesai!

Rata-rata turnaround:

$$\frac{100 + 100 + 110}{3} \approx 103 \text{ detik}$$



Masalah Inti

SJF bersifat **non-preemptive**: sekali A mulai, tidak bisa dihentikan meskipun ada pekerjaan lebih pendek.

STCF: SJF + Preemption

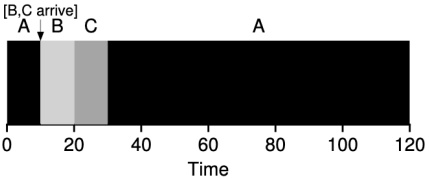
Selalu jalankan pekerjaan dengan sisa waktu terpendek

Ketika B dan C tiba di $t = 10$, penjadwal **mendahului A** dan menjalankan B dan C dulu.

Rata-rata turnaround:

$$\frac{120 + 10 + 20}{3} = 50 \text{ detik}$$

Jauh lebih baik! Preemption memungkinkan penjadwal “menyela” pekerjaan yang sedang berjalan.



STCF = Shortest Time-to-Completion First

Juga dikenal sebagai PSJF (Preemptive SJF). Optimal untuk turnaround time.

Latihan: SJF dan STCF

Uji pemahaman Anda tentang SJF vs STCF

Soal

Tiga proses dengan waktu tiba dan durasi berikut:

- P1: tiba di $t = 0$, durasi = 12 detik
- P2: tiba di $t = 3$, durasi = 5 detik
- P3: tiba di $t = 5$, durasi = 3 detik

Hitung rata-rata turnaround time jika menggunakan:

1. SJF (non-preemptive)
2. STCF (preemptive)

Petunjuk

SJF: proses yang sudah jalan tidak bisa disela. STCF: proses bisa disela jika ada proses baru dengan sisa waktu lebih pendek.

Jawaban: SJF dan STCF

Perbandingan non-preemptive vs preemptive

SJF (non-preemptive):
P1 sudah jalan sejak $t = 0$, tidak bisa disela.

Urutan: P1 (0–12), P3 (12–15), P2 (15–20)

$$\text{Rata-rata} = \frac{12 + 17 + 10}{3} = 13 \text{ dtk}$$

STCF (preemptive):
P1 disela saat proses lebih pendek tiba.

Urutan: P1(0–3), P2(3–8), P3(8–11), P1(11–20)

P	Tiba	Selesai	TA
P1	0	20	20
P2	3	8	5
P3	5	11	6

$$\text{Rata-rata} = \frac{20 + 5 + 6}{3} \approx 10.3 \text{ dtk}$$

Kesimpulan

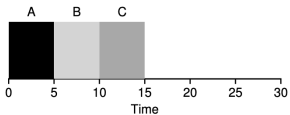
STCF (10.3 dtk) lebih baik dari SJF (13 dtk) karena preemption memungkinkan proses pendek selesai lebih cepat.

Masalah STCF: Response Time Buruk

Optimal turnaround $\neq$ optimal response

Bayangkan 3 pekerjaan tiba bersamaan (A, B, C — masing-masing 5 detik). Dengan SJF:

1. A mulai di $t = 0 \rightarrow$ response = 0
2. B harus tunggu A selesai $\rightarrow$ response = 5
3. C harus tunggu A dan B $\rightarrow$ response = 10



Rata-rata Response Time

$\frac{0+5+10}{3} = 5$ detik. Pengguna di terminal ketiga harus menunggu 10 detik sebelum melihat respons!

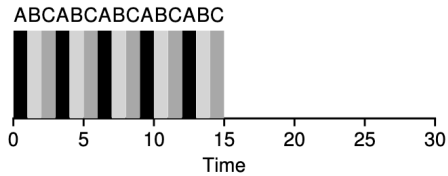
Round Robin: Giliran Berputar

Tiap pekerjaan dapat irisan waktu (time slice) yang sama

RR dengan irisan waktu 1 detik: A, B, C, A, B, C, A, B, C, . . .

Rata-rata response time:

$$\frac{0+1+2}{3} = 1 \text{ detik}$$



Jauh lebih baik dari SJF (5 detik)! Semua proses mendapat giliran **dengan cepat**.

Trade-off RR: Turnaround Time Buruk

Keadilan mengakibatkan penyelesaian lambat

RR meregangkan semua pekerjaan — selesai hampir bersamaan, bukan satu per satu.

Metrik	SJF	RR (ts=1s)
Rata-rata response time	5 detik	**1 detik**
Rata-rata turnaround time	**10 detik**	14 detik

Trade-off Mendasar

Adil (RR) → response bagus, turnaround buruk.
Prioritas pendek dulu (SJF) → turnaround bagus, response buruk.
Tidak ada satu penjadwal yang sempurna untuk keduanya!

Ukuran Time Slice Penting!

Terlalu pendek atau terlalu panjang sama-sama bermasalah

Time slice terlalu pendek

1. Response time sangat baik
2. Tapi overhead context switch tinggi
3. CPU sibuk “administrasi” bukan kerja nyata

Time slice terlalu panjang

1. Overhead rendah
2. Tapi response time memburuk
3. Jika sangat panjang, mirip FIFO!

Prinsip Amortisasi

Pilih time slice cukup panjang agar biaya context switch **teramortisasi**, tapi cukup pendek agar sistem tetap responsif. Contoh: 10–100 ms.

Latihan: Round Robin

Uji pemahaman Anda tentang Round Robin

Soal

Empat proses tiba bersamaan di $t = 0$ dengan durasi: P1 = 6, P2 = 3, P3 = 1, P4 = 4 detik. Time slice = 2 detik.

Dengan algoritma **Round Robin (time slice = 2 detik)**, hitung:

1. Response time masing-masing proses
2. Rata-rata response time
3. Rata-rata turnaround time

Petunjuk

Urutan RR: P1, P2, P3, P4, P1, P2, P4, P1. Jika proses selesai sebelum habis time slice, langsung lanjut ke proses berikutnya.

Jawaban: Round Robin

Pembahasan soal latihan RR

Time slice = 2 detik. Urutan: P1(2), P2(2), P3(1), P4(2), P1(2), P2(1), P4(2), P1(2)

Proses	Durasi	Mulai	Response	Selesai	Turnaround
P1	6	0	0	14	14
P2	3	2	2	11	11
P3	1	4	4	5	5
P4	4	5	5	13	13

$$\text{Rata-rata response} = \frac{0+2+4+5}{4} = 2.75 \text{ detik}$$

$$\text{Rata-rata turnaround} = \frac{14+11+5+13}{4} = 10.75 \text{ detik}$$

Perhatikan Trade-off!

Response time sangat baik (2.75 dtk), tapi turnaround buruk (10.75 dtk) — semua proses selesai “hampir bersamaan” karena RR meregangkan eksekusi.

Ringkasan Algoritma Dasar

Perbandingan empat algoritma yang sudah dibahas

Algoritma	Turnaround	Response	Catatan
FIFO	Buruk (convoy)	Buruk	Sangat sederhana
SJF	**Optimal***	Buruk	Non-preemptive
STCF	**Optimal**	Buruk	Preemptive SJF
RR	Buruk	**Baik**	Adil, time-slicing

* Optimal jika semua tiba bersamaan

Pertanyaan Besar

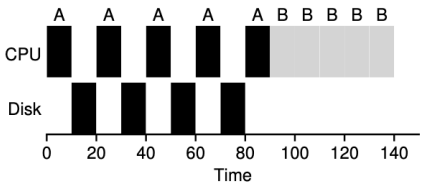
Bisakah kita membangun penjadwal yang bagus di **kedua** metrik, tanpa harus tahu durasi pekerjaan?

Pekerjaan Tidak Selalu Hanya Pakai CPU

Menangani proses yang melakukan I/O

Contoh: Job A (50 ms CPU, tapi setiap 10 ms melakukan I/O 10 ms) dan Job B (50 ms CPU, tanpa I/O).

Tanpa optimasi: CPU menganggur saat A menunggu I/O.



Masalah

Jika penjadwal tidak bijak, CPU akan **menganggur** sia-sia saat proses menunggu I/O!

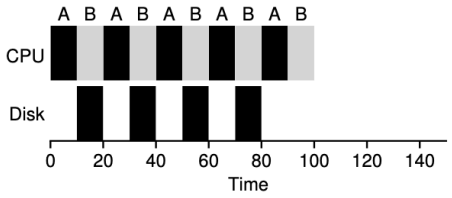
Overlap: Gunakan CPU Saat Proses Lain Menunggu I/O

Tumpang tindih kerja CPU dan I/O

Solusi: Perlakukan setiap “burst” CPU sebagai sub-pekerjaan independen.

Saat A menunggu I/O, B menggunakan CPU. Saat I/O A selesai, A kembali mendahului B (STCF).

$\rightarrow$ CPU dan disk keduanya sibuk = **utilisasi tinggi**!



Prinsip Overlap

Tumpang tindihkan operasi CPU dan I/O untuk memaksimalkan pemanfaatan sumber daya.

Apa yang Masih Kurang?

Mengapa algoritma-algoritma tadi belum cukup

1. SJF/STCF butuh tahu **durasi pekerjaan** $\rightarrow$ di dunia nyata, ini **mustahil**!
2. RR bagus untuk response, tapi turnaround buruk
3. Bagaimana kalau kita ingin **keduanya** tanpa pengetahuan *a priori*?

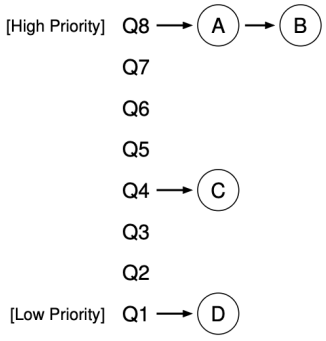
Jawabannya: MLFQ

Multi-Level Feedback Queue (MLFQ) = penjadwal yang **belajar dari perilaku** proses untuk membuat keputusan yang cerdas.

Ide Utama MLFQ

Belajar dari perilaku masa lalu untuk memprediksi masa depan

1. Banyak antrian dengan tingkat **prioritas** berbeda
2. Proses baru dianggap “pendek” $\rightarrow$ prioritas tinggi
3. Jika proses terus pakai CPU lama $\rightarrow$ prioritas turun
4. Jika proses sering lepas CPU (I/O) $\rightarrow$ prioritas tetap tinggi



Filosofi MLFQ

Tidak perlu tahu durasi pekerjaan! MLFQ mengamati perilaku dan **menyesuaikan** prioritas secara otomatis.

Dua Aturan Dasar MLFQ

Rules 1 & 2: Prioritas menentukan segalanya

Aturan 1

Jika Priority(A) $>$ Priority(B), maka A berjalan (B tidak).

Aturan 2

Jika Priority(A) $=$ Priority(B), maka A dan B bergantian dalam mode **Round Robin**.

Pertanyaan kunci: Bagaimana prioritas ini ditentukan? $\rightarrow$ Jawabannya: dari **perilaku** proses!

Analogi

Bayangkan antrian loket: VIP (prioritas tinggi) dilayani duluan. Tapi siapa yang jadi VIP? Yang membutuhkan layanan singkat!

Bagaimana Prioritas Berubah?

Rules 3, 4a, 4b: Mekanisme naik-turun antrian

Aturan 3

Proses baru selalu masuk di antrian **paling atas** (prioritas tertinggi).

Aturan 4a

Jika proses menghabiskan seluruh jatah waktunya $\rightarrow$ prioritas **turun** (turun satu antrian).

Aturan 4b

Jika proses melepas CPU sebelum jatah habis (misalnya I/O) $\rightarrow$ prioritas **tetap**.

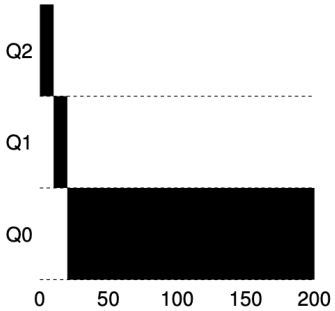
Logika: Proses yang pakai CPU lama = “batch job” $\rightarrow$ turun. Proses yang sering lepas CPU = “interaktif” $\rightarrow$ tetap tinggi.

Contoh 1: Pekerjaan Berjalan Lama

Proses CPU-bound turun dari Q2 ke Q0

Sebuah proses CPU-bound masuk di Q2 (tertinggi).

1. Habiskan time slice di Q2 $\rightarrow$ turun ke Q1
2. Habiskan time slice di Q1 $\rightarrow$ turun ke Q0
3. Bertahan di Q0 selamanya (prioritas terendah)



Interpretasi

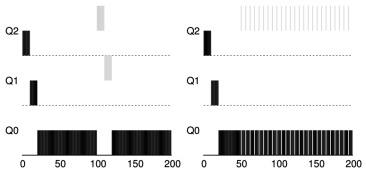
MLFQ “menghukum” proses yang terus-menerus makan CPU. Ini memberi jalan kepada proses pendek dan interaktif.

Contoh 2: Pekerjaan Pendek Muncul

MLFQ mendekati perilaku SJF secara otomatis

A (CPU-bound, panjang) sudah di Q0. Lalu B (pendek, 20 ms) tiba.

1. B masuk di Q2 (tertinggi) $\rightarrow$ langsung jalan
2. B selesai dalam 2 time slice
3. A melanjutkan di Q0 setelah B selesai



Seperti SJF!

Tanpa tahu durasi B, MLFQ secara otomatis **memprioritaskan pekerjaan pendek**. Ini terjadi karena proses baru selalu masuk di atas.

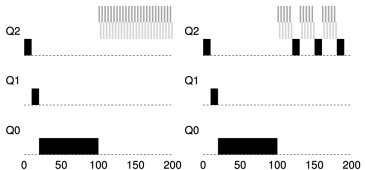
Contoh 3: Proses Interaktif dengan I/O

Proses I/O-intensif tetap di prioritas tinggi

B (interaktif): pakai CPU hanya 1 ms lalu lepas untuk I/O.

Karena B melepas CPU sebelum jatah habis (Aturan 4b), MLFQ menjaga B di prioritas **tertinggi**.

$\rightarrow$ Pengguna yang mengetik di terminal mendapat respons cepat!



Kesimpulan

MLFQ otomatis mengenali proses interaktif dan memberi mereka layanan **cepat**.

Tiga Masalah MLFQ Versi Awal

Kelemahan serius yang perlu diatasi

1. **Starvation (Kelaparan):** Jika ada banyak proses interaktif, proses CPU-bound di Q0 **tidak pernah** mendapat giliran.
2. **Gaming the Scheduler (Pemermainan):** Proses nakal bisa sengaja melepas CPU tepat sebelum jatah habis (misalnya I/O ke file kosong), sehingga tetap di prioritas tinggi dan memonopoli CPU hingga 99%!
3. **Perubahan Perilaku:** Proses yang awalnya CPU-bound lalu menjadi interaktif tidak akan naik prioritas lagi.

Kesimpulan

Aturan awal MLFQ mudah dieksploitasi dan tidak adil untuk proses jangka panjang.

Solusi 1: Priority Boost (Aturan 5)

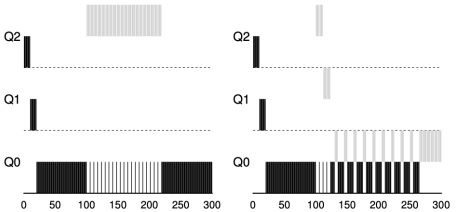
Secara berkala, naikkan semua proses ke antrian teratas

Aturan 5

Setelah periode waktu *S*, **pindahkan semua** proses ke antrian paling atas.

Manfaat:

1. Proses CPU-bound dijamin **tidak kelaparan**
2. Proses yang berubah perilaku mendapat **kesempatan baru**



Kiri vs Kanan

Kiri: tanpa boost, proses panjang kelaparan. Kanan: dengan boost tiap *S* ms, proses panjang tetap mendapat giliran.

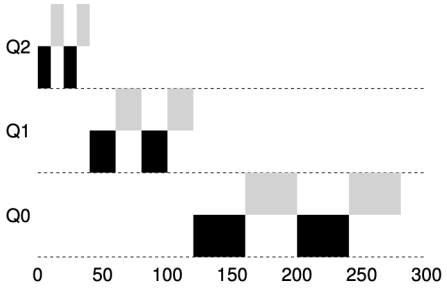
Solusi 2: Better Accounting (Aturan 4 Baru)

Lacak total waktu CPU, bukan per burst

Aturan 4 (Revisi)

Setelah proses menggunakan **total** jatah waktunya pada satu tingkat (berapa kali pun ia melepas CPU), prioritas diturunkan.

Jadi proses tidak bisa “curang” dengan I/O tepat sebelum jatah habis — total CPU time-nya tetap dihitung!



Kiri vs Kanan

Kiri: proses gaming memonopoli CPU. Kanan: akuntansi akurat mencegah pemermainan.

Lima Aturan Final MLFQ

Kumpulan aturan yang sudah disempurnakan

1. **Rule 1:** Jika Priority(A) $>$ Priority(B), A berjalan
2. **Rule 2:** Jika Priority(A) $=$ Priority(B), A & B dalam Round Robin
3. **Rule 3:** Proses baru masuk di antrian **paling atas**
4. **Rule 4:** Setelah total jatah waktu habis di satu level, proses **turun**
5. **Rule 5:** Setiap periode S, semua proses **dinaikkan** ke atas

Kenapa MLFQ Hebat?

Tanpa tahu apa-apa tentang proses, MLFQ mengamati perilaku dan membuat keputusan yang mendekati **optimal** untuk turnaround DAN response time!

Latihan: MLFQ

Uji pemahaman Anda tentang Multi-Level Feedback Queue

Soal

Sebuah sistem MLFQ memiliki 3 antrian (Q2 tertinggi, Q0 terendah) dengan time slice = 10 ms per level. Dua proses:

- **Proses A**: CPU-bound, butuh 40 ms total CPU
- **Proses B**: interaktif, setiap 3 ms melakukan I/O (melepas CPU)

Keduanya tiba bersamaan di $t = 0$.

Pertanyaan:

1. Di antrian mana A berada setelah 30 ms?
2. Di antrian mana B berada setelah 30 ms? Mengapa?
3. Apakah B perlu menunggu A? Jelaskan!

Jawaban: MLFQ

Pembahasan soal latihan MLFQ

Proses A (CPU-bound, 40 ms total):

1. $t = 0$–10: jalan di Q2, habiskan time slice $\rightarrow$ turun ke Q1
2. $t = 10$–20: jalan di Q1, habiskan time slice $\rightarrow$ turun ke Q0
3. Setelah 30 ms: A ada di **Q0** (prioritas terendah)

Proses B (interaktif, I/O setiap 3 ms):

1. B selalu melepas CPU sebelum jatah 10 ms habis (Aturan 4b)
2. Setelah 30 ms: B **tetap di Q2** (prioritas tertinggi)

B tidak perlu menunggu A!

Karena B di Q2 dan A di Q0, B selalu diprioritaskan (Aturan 1). Ini menunjukkan MLFQ secara otomatis memberikan layanan cepat untuk proses **interaktif**.

Parameter yang Perlu Di-tuning

Konstanta voo-doo dalam MLFQ

Parameter	**Dampak**
Jumlah antrian	Lebih banyak = granularitas prioritas lebih halus
Time slice per antrian	Atas: pendek (10 ms). Bawah: panjang (100+ ms)
Periode boost (S)	Terlalu besar: starvation. Terlalu kecil: kurang efisien
Jatah waktu (allotment)	Berapa lama sebelum turun level

Hukum Ousterhout

Hindari “konstanta ajaib” yang sulit ditentukan. Tapi di MLFQ, tuning parameter tetap diperlukan — biasanya menggunakan **default** yang sudah terbukti baik.

MLFQ di OS Nyata

Implementasi pada Solaris, BSD, dan Windows

Solaris

1. 60 antrian prioritas
2. Time slice: 20ms (atas) sampai ratusan ms (bawah)
3. Boost setiap $\sim$1 detik
4. Konfigurasi via tabel admin

BSD / Windows NT

1. Menggunakan formula decay-usage
2. Prioritas meluruh seiring penggunaan CPU
3. User bisa atur via `nice` (Unix) atau Task Manager priority

Fakta Penting

Hampir semua OS modern — Linux, macOS, Windows — menggunakan varian **MLFQ** sebagai penjadwal dasar.

Perbandingan Lengkap Semua Algoritma

Dari yang paling sederhana ke yang paling cerdas

Algoritma	Turn-around	Response	Pre-empt	Perlu tahu durasi	Masalah utama
FIFO	Buruk	Buruk	Tidak	Tidak	Convoy effect
SJF	Optimal*	Buruk	Tidak	Ya	Non-preemptive
STCF	Optimal	Buruk	Ya	Ya	Butuh oracle
RR	Buruk	Baik	Ya	Tidak	Turnaround buruk
MLFQ	Baik	Baik	Ya	**Tidak**	Tuning parameter

MLFQ = Jawaban Praktis

MLFQ tidak perlu tahu durasi, bersifat preemptive, dan mendekati optimal untuk kedua metrik!

Ringkasan Konseptual

Poin inti yang harus diingat

Lima Kalimat Kunci

Penjadwalan adalah soal memilih proses mana yang dijalankan berikutnya oleh CPU, dengan trade-off antara kecepatan penyelesaian dan kecepatan respons.

1. FIFO sederhana tapi rentan convoy effect
2. SJF/STCF optimal untuk turnaround, tapi butuh tahu durasi
3. Round Robin bagus untuk response, buruk untuk turnaround
4. MLFQ menggabungkan keduanya tanpa perlu tahu durasi
5. Kunci MLFQ: belajar dari perilaku, bukan menebak masa depan

Soal Latihan Mandiri (1/2)

Kerjakan dan kuasai semua soal ini!

Perhatian!

Setiap mahasiswa **wajib menguasai** semua soal berikut. Minggu depan, beberapa mahasiswa akan dipilih secara **acak** untuk maju ke depan dan menjelaskan jawabannya.

Soal 1 — FIFO vs SJF

Empat proses tiba bersamaan di $t = 0$: P1 = 15 dtk, P2 = 3 dtk, P3 = 7 dtk, P4 = 1 dtk. Hitung rata-rata turnaround time untuk FIFO (urutan P1, P2, P3, P4) dan SJF. Berapa persen perbaikan SJF dibanding FIFO?

Soal 2 — STCF

P1 tiba di $t = 0$ (durasi 10 dtk), P2 tiba di $t = 2$ (durasi 4 dtk), P3 tiba di $t = 4$ (durasi 2 dtk). Gambarkan timeline eksekusi STCF dan hitung rata-rata turnaround time.

Soal Latihan Mandiri (2/2)

Kerjakan dan kuasai semua soal ini!

Soal 3 — Round Robin

Tiga proses tiba di $t = 0$: P1 = 9 dtk, P2 = 6 dtk, P3 = 3 dtk. Time slice = 3 detik.
Hitung: (a) rata-rata response time, (b) rata-rata turnaround time. Bandingkan hasilnya dengan SJF — algoritma mana yang lebih baik untuk masing-masing metrik?

Soal 4 — MLFQ

Sebuah MLFQ memiliki 3 antrian (Q2, Q1, Q0) dengan time slice 8 ms per level dan priority boost setiap $S = 50$ ms.
Proses X (CPU-bound, butuh 100 ms) dan Y (interaktif, I/O setiap 5 ms) tiba bersamaan.
(a) Di antrian mana X dan Y berada setelah 30 ms?
(b) Apa yang terjadi saat boost di $t = 50$ ms? Mengapa ini penting?

Ingat!

Pahami **logika** di balik jawaban, bukan hanya hasilnya. Anda harus bisa **menjelaskan** ke depan kelas!

Penutup

Arah belajar lanjutan setelah Week 3

Lanjut Bacaan

Untuk memperkuat konsep, baca ulang bab penjadwalan di buku terjemahan OSTEP dan coba simulasi `scheduler.py` dan `mlfq.py`.

1. `chapter4-book-translate.tex`: FIFO, SJF, STCF, RR
2. `chapter5-book-translate.tex`: MLFQ dan tuningnya
3. Coba simulator: `scheduler.py` dan `mlfq.py` dari repo OSTEP