01.1. Підходи до планування паралельного виконання ОП

Три аспекти:

1. Організація розподілу процесорного часу з точки зору забезпечення багатозадачності операційного середовища;

2. Алгоритми планування паралельного виконання процесів (scheduling disciplines);

3. Програмна реалізація диспетчера (як правило, на основі комбінації декількох алгоритмів планування) → ефективність реалізації («швидкі» структури даних, черга → {active queue, expired queue})

01.2. Підходи до планування паралельного виконання ОП

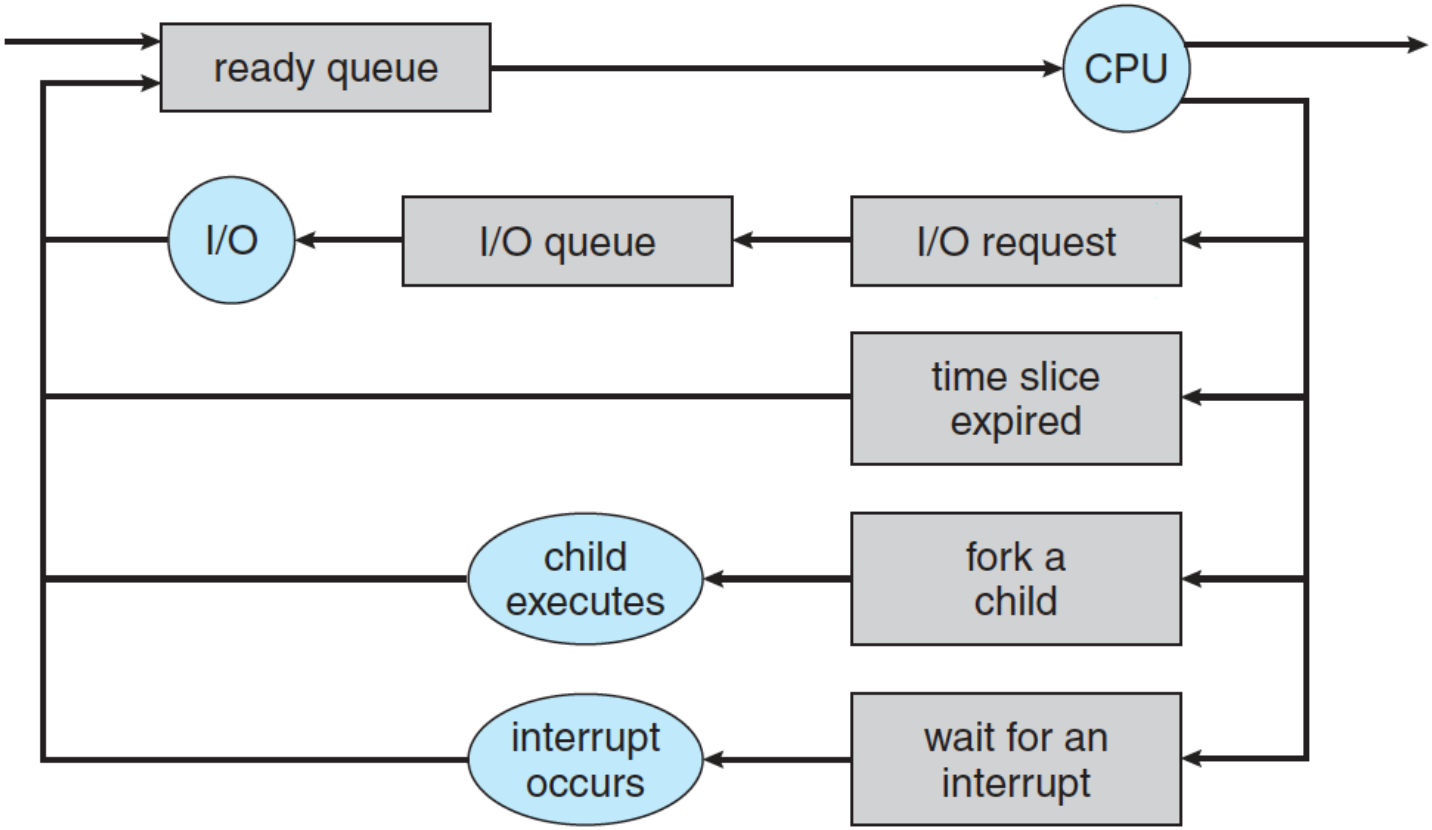
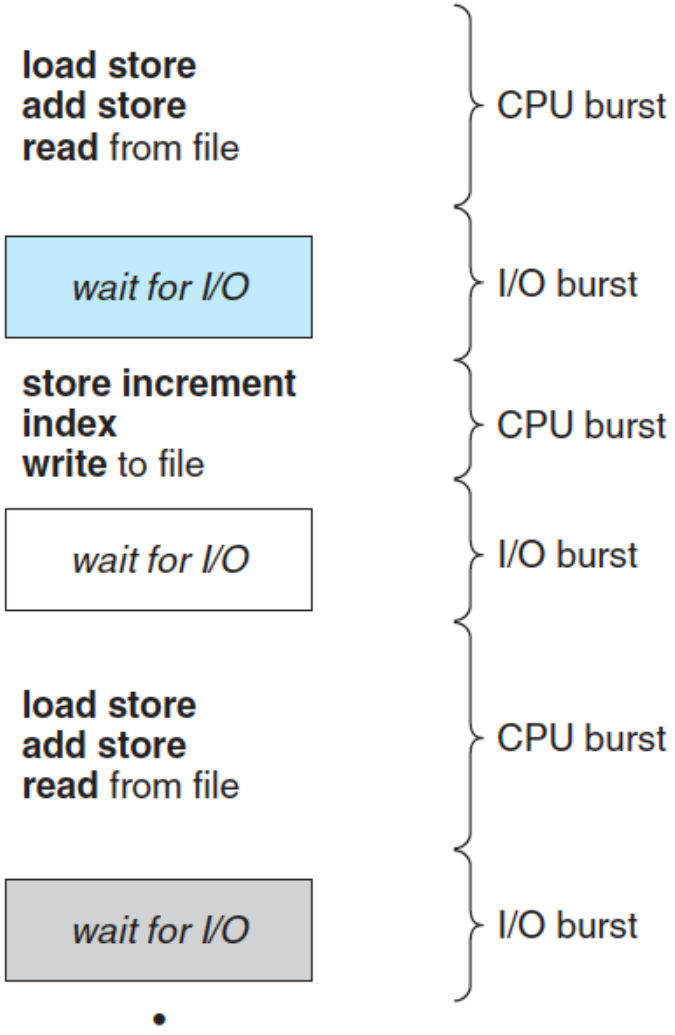


Схема використання черг готовності та очікування

01.3. Підходи до планування паралельного виконання ОП



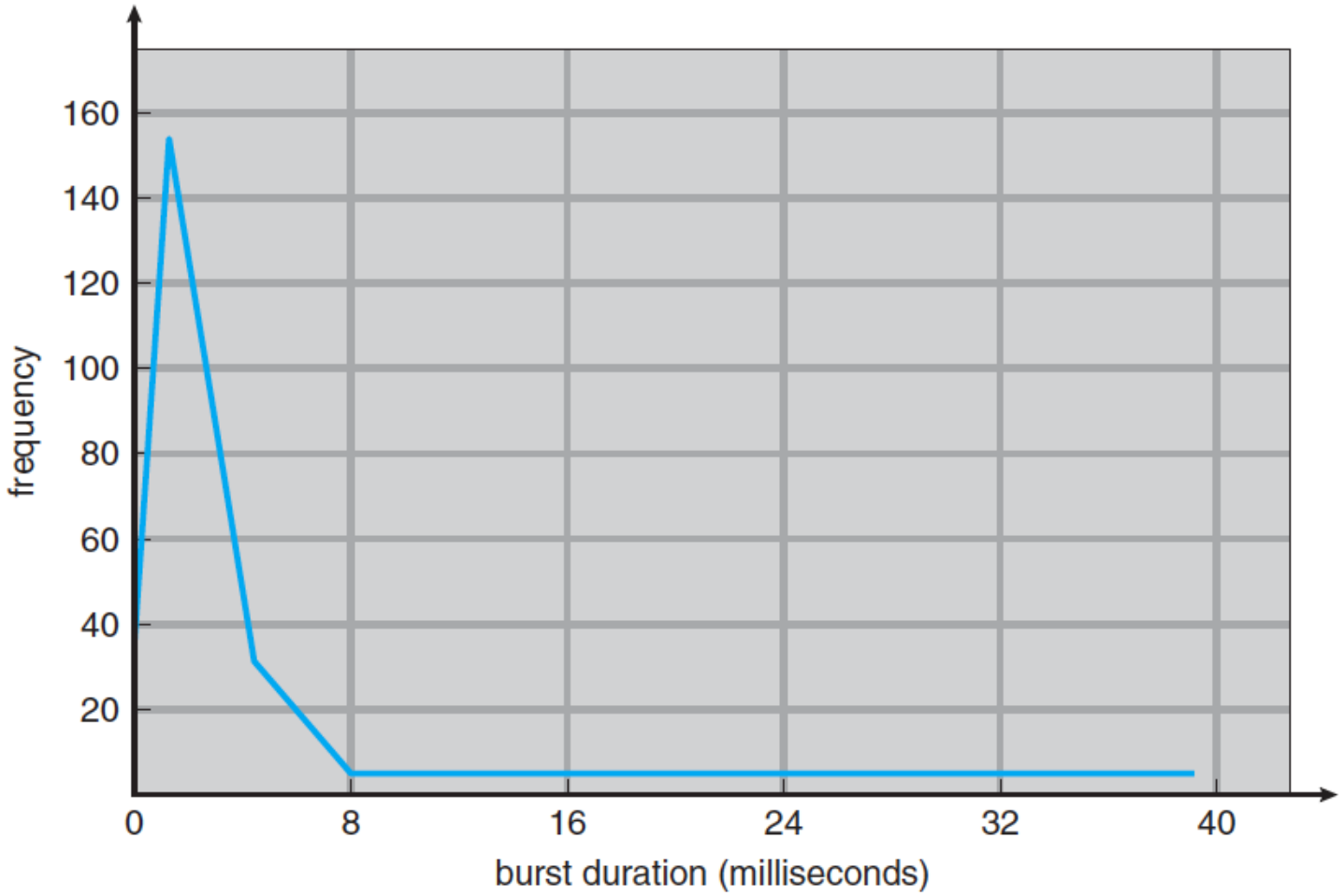
Під час виконання ОП почергово знаходиться у двох станах:

1) виконання у процесорі (**CPU burst**);
2) очікування операції вводу/виводу (**I/O burst**).

burst = «спалах» [активності]

Дослідження: виміри величини CPU burst для різних процесів та різних процесорних архітектур

01.4. Підходи до планування паралельного виконання ОП



Типова гістограма тривалості CPU burst

01.5. Підходи до планування паралельного виконання ОП

- переважна більшість CPU burst є короткими;
- кількість тривалих CPU burst мала;
- I/O-bound процес, як правило, має велику кількість коротких CPU burst;
- CPU-bound процес може мати декілька тривалих CPU burst;
- інформація про величину CPU burst використовується при виборі та розробці алгоритмів диспетчеризації у сучасних ОС.

01.6. Підходи до планування паралельного виконання ОП

4-ри ситуації, в яких може прийматися рішення по плануванню:

1. Перехід процеса зі стану «Виконання» у стан «Очікування» (наприклад, внаслідок запиту операції вводу/виводу або виклику wait());
2. Перехід процеса зі стану «Виконання» у стан «Готовність» (наприклад, внаслідок виникнення переривання);
3. Перехід процеса зі стану «Очікування» у стан «Готовність» (наприклад, після завершення операції вводу/виводу);
4. Завершення виконання процеса.

02.1. Планування паралельного виконання ОП

1) багатозадачність **без витіснення** (non-preemptive or cooperative multitasking) → лише ситуації 1 і 4 → рішення про звільнення процесора приймає сам процес (обов'язки по прийняттю рішення про звільнення процесора частково або повністю делеговані процесам);

2) багатозадачність **з витісненням** (preemtive multitasking) → ситуації 1,2,3,4 → рішення про звільнення процесора приймає диспетчер процесів (всі обов'язки по прийняттю рішення про звільнення процесора бере на себе ядро ОС (=«центр»));

02.2. Планування паралельного виконання ОП

- Співвідношення цих підходів визначає ступінь централізації механізму планування паралельного виконання процесів.

- Архітектурні обмеження деяких систем допускають тільки багатозадачність без витіснення (cooperative multitasking). Наприклад, системи без таймера.

- Багатозадачність з витісненням (preemtive multitasking) породжує низку проблем, пов’язаних з 1) одночасним доступом до даних процесів, контексти яких переключаються; 2) станом структур даних ядра, які «обслуговують» процес, який витісняється з CPU.

02.3. Планування паралельного виконання ОП

Основні задачі:
1) визначення моменту часу заміни виконуваного процеса іншим;
2) вибір процесу для виконання з черги готових процесів;
3) переключення контекстів «попереднього» та «наступного» процесів.

- **система з розділенням у часі** (час процесора поділяється між процесами) → головний ресурс: процесорний час
- основна вимога: **«справедливий» розподіл ресурсів** між процесами + забезпечення прийнятної швидкості реакції на «зовнішні» події (в тому числі дії користувача)

02.4. Планування паралельного виконання ОП

Алгоритми планування:

1. На основі квантування.

2. На основі пріоритетів.

Квантування:

- процесам відводяться часові кванти на виконання;
- квант: від 10 млс до 100 млс;
- варіанти: однакові / не однакові, фіксовані / змінні;
- черга: циклічна (round robin), FIFO, LIFO, ...

02.5. Планування паралельного виконання ОП

Пріоритети:

- чим менше (більше) число, тим вище пріоритет (“важливість”, перевага над іншими) процесу;
- варіанти: данамічний / статичний пріоритет;
- відносні пріоритети: процес виконується доти, доки сам не звільнить процесор;
- абсолютні пріоритети: процес виконується доти, доки в черзі не з’явиться процес з більшим пріоритетом.

Приклад: в системі UNIX реалізована комбінована схема (кванти + пріоритети).

02.6. Алгоритми планування (Scheduling algorithms)

1. First-Come, First-Served (FCFS)
2. Shortest-Job-First (SJF)
3. Priority scheduling
4. Round-Robin scheduling
5. Multilevel Queue scheduling
6. Multilevel Feedback Queue Scheduling

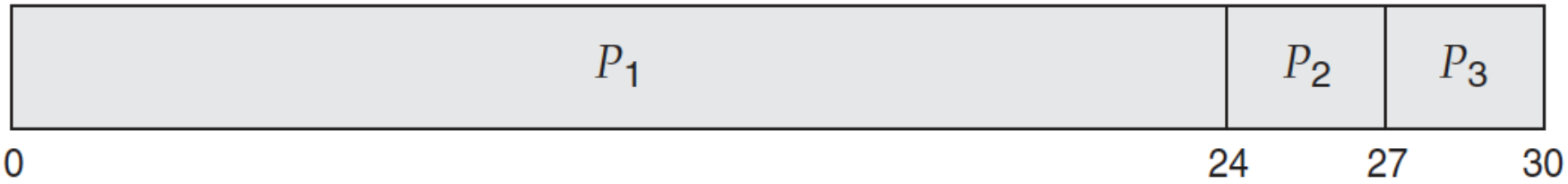
03.1. Алгоритми планування: First-Come, First-Served (FCFS)

- «перший прийшов - перший обслуговується»: FIFO queue (FIFO черга);
- тип багатозадачності: без витіснення (nonpreemptive);
- проста реаліація: коли процес надходить до черги готовності, його PCB додається в кінець («хвіст») відповідного зв’язаного списку;
- коли CPU звільняється, він надається першому у списку (=черзі) процесу, PCB якого видаляється з «голови» черги.

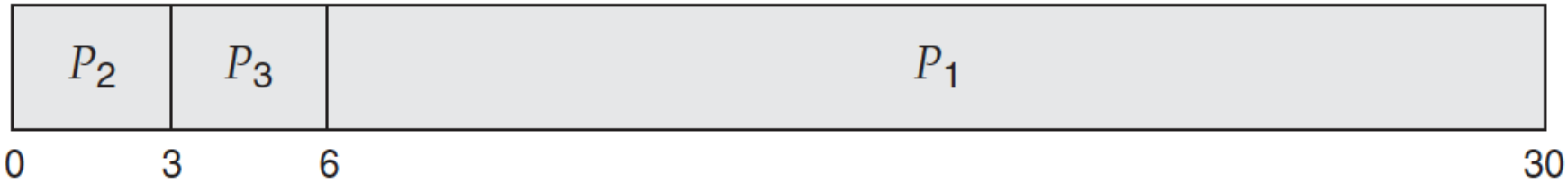
03.2. Алгоритми планування: First-Come, First-Served (FCFS)

Process	Burst Time
P_1	24
P_2	3
P_3	3

Діаграми Ганта (Gantt chart) для процесів P_1, P_2, P_3



середній час очікування: (0 + 24 + 27)/3 = 17 млс



середній час очікування: (6 + 0 + 3)/3 = 3 млс

03.3. Алгоритми планування: First-Come, First-Served (FCFS)

Висновки:

1. FCFS не забезпечує мінімум середнього часу очікування, який може коливатись в значних межах в залежності від величини CPU burst процесів.

2. Основною перевагою FCFS є дуже проста реалізація алгоритму.

04.1. Алгоритми планування: Shortest-Job-First (SJF)

- першим виконується найкоротше завдання
- кожному процесу ставиться у відповідність очікувана величина його наступного CPU burst
- коли CPU звільняється, він надається процесу з найменшою величиною наступного CPU burst
- якщо в двох чи більше процесів однаковий найменший CPU burst, то для них застосовується FCFS
- доведено, що з точки зору мінімізації середнього часу очікування, **SJF є оптимальним алгоритмом**

04.2. Алгоритми планування: Shortest-Job-First (SJF)

- Основна проблема: визначення очікуваної величини наступного CPU burst
- Наближений SJF → прогнозування величини CPU burst
- Один з найбільш розповсюджених і найпростіших методів прогнозування: метод **експоненціального усереднення** (exponential average), який виконує зваження по давнині

04.3. Алгоритми планування: Shortest-Job-First (SJF)

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$$

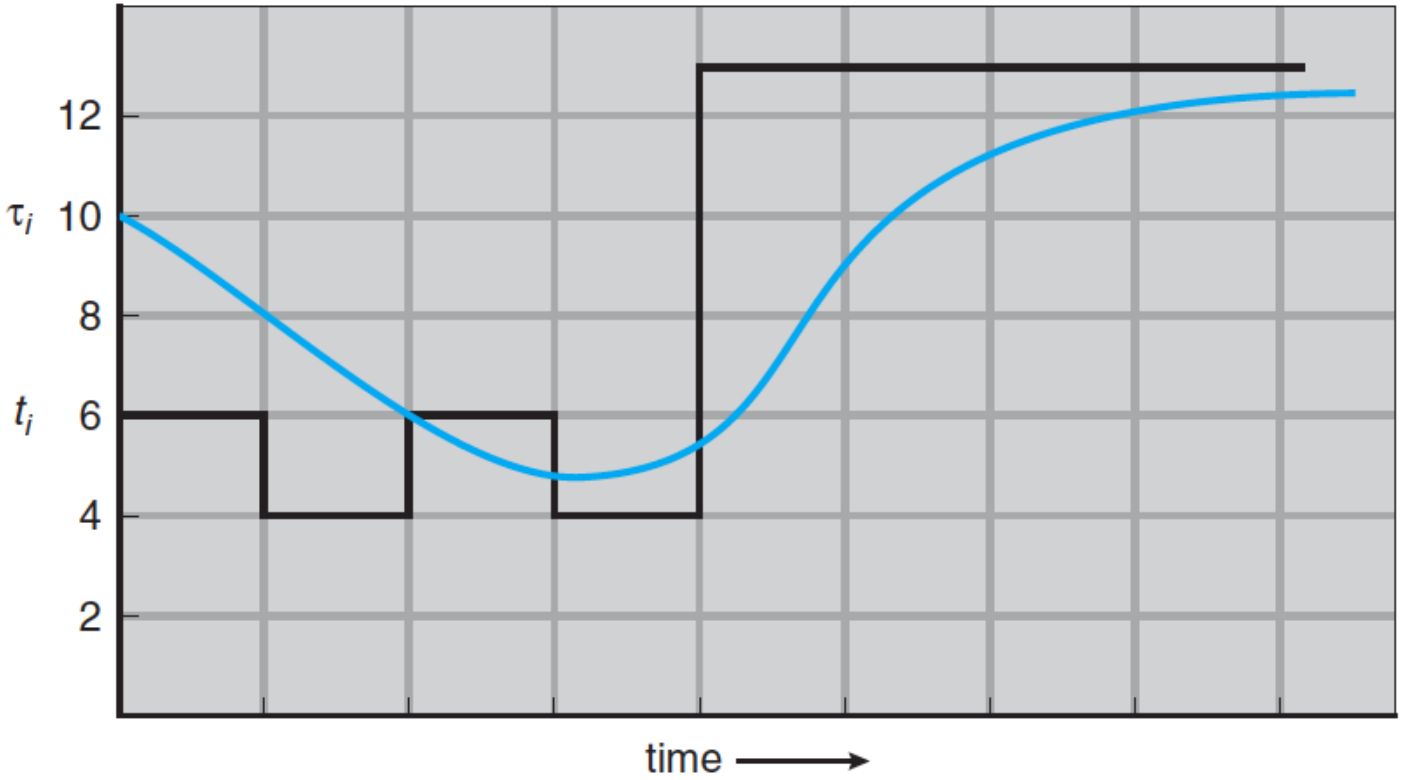
t_n — величина CPU burst процеса на n-му кроці (=«теперішній час» = «дійсність»);

τ_n — величина CPU burst спрогнозована на n-му кроці (=«досвід минулого»);

τ_{n+1} — прогноз на наступний n+1-ий крок;

α — ваговий коефіцієнт: $0 \le \alpha \le 1$, яким визначається скільки ваги приділяти «теперішньому» (t_n) в противагу «минулому» (τ_n).

04.4. Алгоритми планування: Shortest-Job-First (SJF)



$\alpha = 1/2$

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

04.5. Алгоритми планування: Shortest-Job-First (SJF)

Тип багатозадачності: 1) без витіснення; 2) з витісненням.

В момент надходження в чергу готовності нового процесу, його очікуваний CPU burst може бути меншим ніж час, який лишився для виконання процеса, що захопив CPU.

1) без витіснення: процесу в CPU дається можливість завершити свій CPU burst;

2) з витісненням: поточний процес витісняється з CPU, а його місце займає процес з меншим очікуваним CPU burst → shortest-remaining-time-first (процес з найменшим часом, який лишився для виконання, виконується першим).

05.1. Алгоритми планування: Priority scheduling

- кожному процесу ставиться у відповідність пріоритет
- коли CPU звільняється, він надається процесу з найбільшим пріоритетом
- якщо в двох чи більше процесів однаковий пріоритет, то для них застосовується FCFS
- SJF можна розглядати, як частковий випадок Priority scheduling
- значення пріоритету задається цілим числом
- ОС Linux: чим менше число, тим більше пріоритет

05.2. Алгоритми планування: Priority scheduling

Тип багатозадачності: 1) без витіснення; 2) з витісненням.

В момент надходження в чергу готовності нового процесу, його пріоритет може бути більшим ніж пріоритет процеса, що захопив CPU.

1) без витіснення: процесу в CPU дається можливість завершити свій CPU burst;

2) з витісненням: поточний процес витісняється з CPU новим процесом з більшим пріоритетом

05.3. Алгоритми планування: Priority scheduling

Значення пріоритету може:

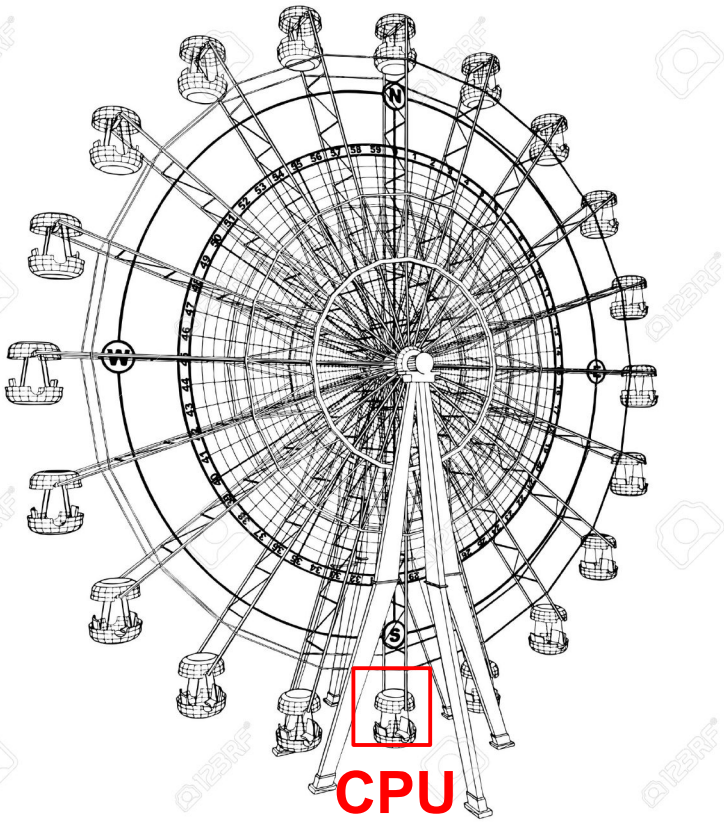
1) визначатись «внутрішньо» на основі величин, які характеризують виконання процесу (загальний час виконання, об’єм потрібної процесу пам’яті, кількість відкритих файлів, час проведений в черзі очікування, відношення середнього I/O burst до середнього CPU burst);

2) призначатись «ззовні», виходячі з «важливості» процесу, розміру оплати процесорного часу користувачем-власником процесу (web-hosting) і т.п.

05.4. Алгоритми планування: Priority scheduling

- Основна проблема: можливість «вічного» блокування (**indefinite blocking** = starvation) процеса з низьким пріоритетом, тобто виникнення ситуації, коли не забезпечується «справедливий» розподіл ресурсу (CPU) між процесами
- Рішення цієї проблеми отримало назву «старіння» (aging)
- **«Старіння» (aging)**: поступове збільшення пріоритету процеса, який довго знаходиться в черзі готовності
- Наприклад в ядрі ОС Linux (v.2.4) O(N) scheduler при кожному спрацюванні збільшує динамічний пріоритет таких процесів на 1-ин

06.1. Алгоритми планування: Round-Robin scheduling



Round-Robin (RR) розроблений спеціально для систем з розділенням в часі.

RR це FCFS з доданим витісненням процесу з CPU після закінчення виділеного йому на роботу відрізку часу - **квант часу (time quantum)** = відрізок часу (time slice).

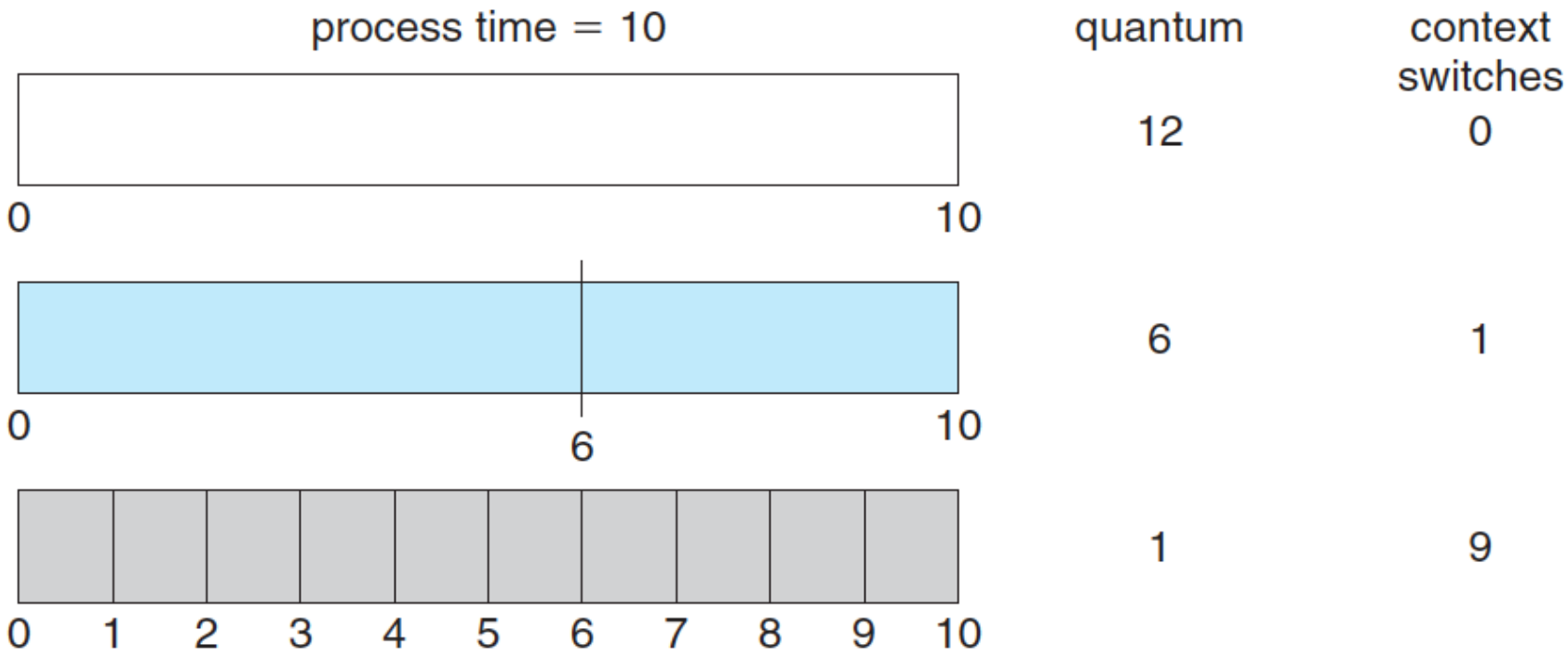
06.2. Алгоритми планування: Round-Robin scheduling

- Черга готовності (ready queue) → кільцева черга (circular queue)
- Коли процес завершується сам, або закінчується виділений йому час, він встає в «хвіст» кільцевої черги, а його місце в CPU займає процес з «голови» кільцевої черги.
- Час очікування в черзі напряму залежить від кількості процесів: чим їх більше, тим більше час очікування процесом наступного кванта часу.

06.3. Алгоритми планування: Round-Robin scheduling

- Продуктивність RR залежить від розміру кванта часу.
- Один крайній випадок: дуже великий розмір кванта → RR = FCFS.
- Другий крайній випадок: дуже малий розмір кванта → занадто багато переключень контексту (на які буде витрачатись більше часу, ніж на виконання процесів).
- В сучасних системах: квант = 10-100 млс, час переключення контексту = 10 мкс (<0,01 кванта).
- Евристичне правило: 80% всіх CPU burst має бути менше кванта.

06.4. Алгоритми планування: Round-Robin scheduling



Приклад: процес, на виконання якого потрібно 10 одиниць часу

07.1. Алгоритми планування: Multilevel Queue scheduling

- Алгоритм багаторівневої черги (Multilevel Queue) створений для випадків, коли процеси можна легко розділити на різні групи.

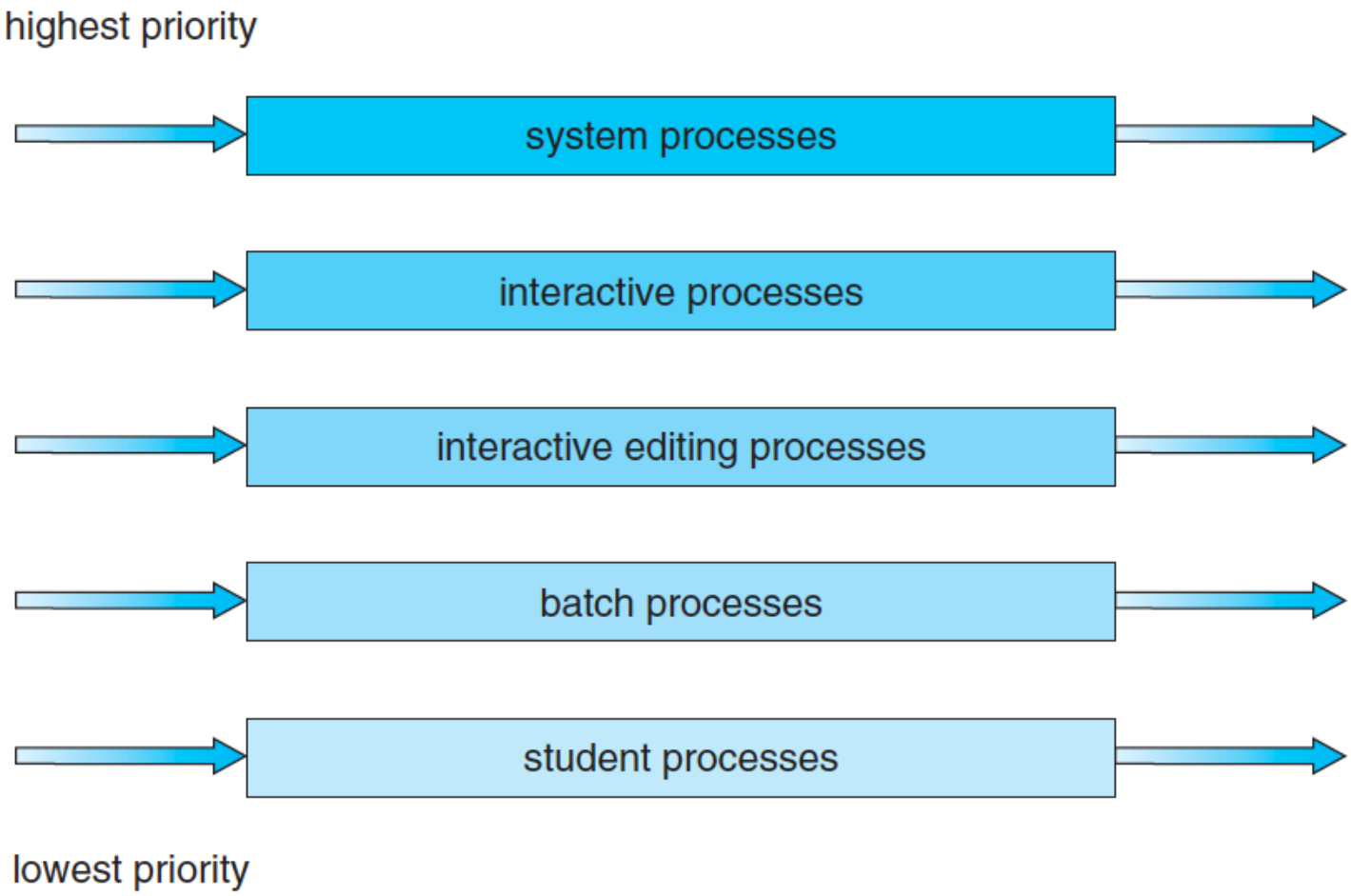
- Приклад: інтерактивні (foreground) та фонові (background) процеси → мають різні вимоги до часу відгуку (response-time), тому для них можна застосувати різні алгоритми планування.

- Черга готовності розділяється на **декілька окремих черг**, для кожної з яких застосовується свій алгоритм планування.

07.2. Алгоритми планування: Multilevel Queue scheduling

- Процеси розділяються між чергами на основі якихось своїх ознак чи параметрів.
- Додатково виконуєтья диспетчеризація черг: fixed-priority preemptive scheduling (витіснення з фіксованим пріоритетом).

07.3. Алгоритми планування: Multilevel Queue scheduling



08.1. Алгоритми планування: Multilevel Feedback Queue Scheduling

- Багаторівнева черга зі зворотнім зв’язком (Multilevel Feedback Queue).
- Процесам дозволяється переміщатися між різними чергами (на відміну від «звичайної» Multilevel Queue).
- Основна ідея: розділяти процеси на групи по ходу їх виконання на основі характеристик їх CPU burst (які можуть змінюватись в часі → адаптація до змін).

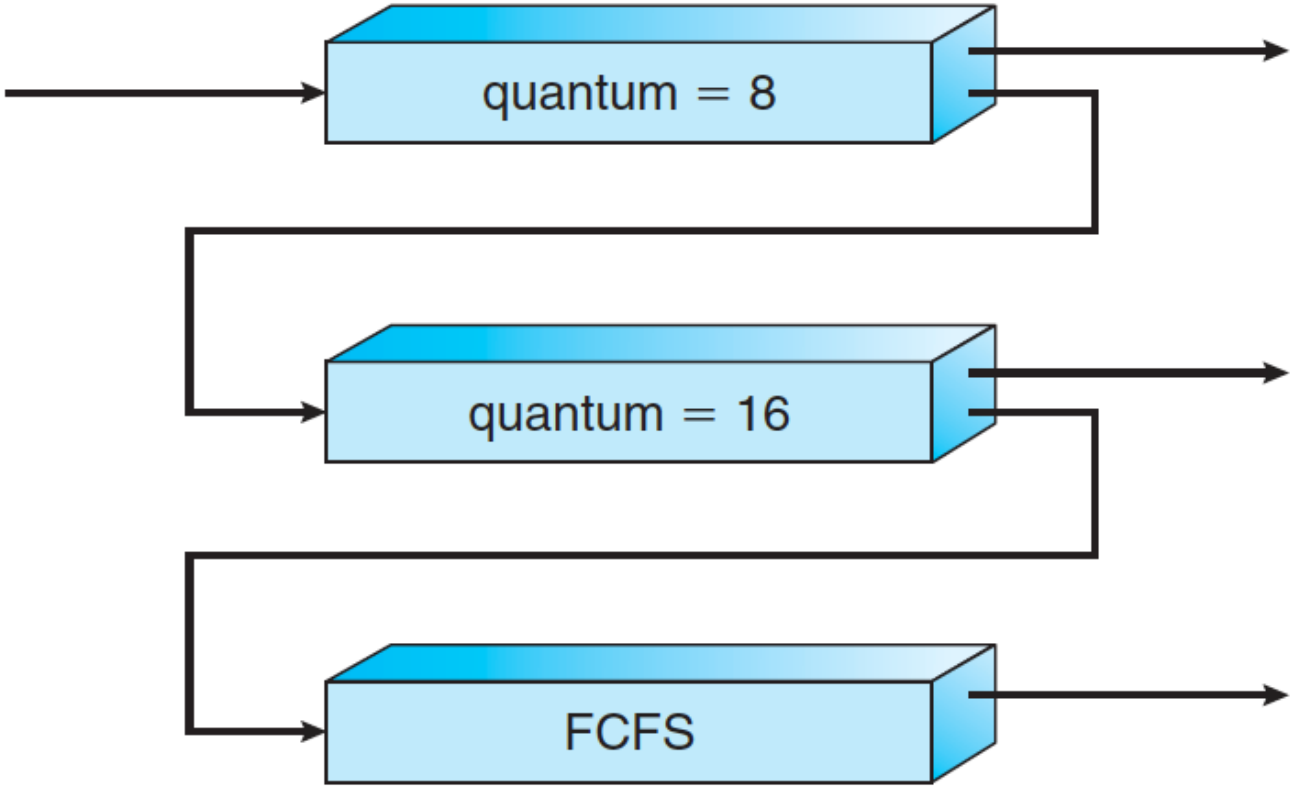
08.2. Алгоритми планування: Multilevel Feedback Queue Scheduling

1. Процеси, які споживають багато процесорного часу «спускаються» в черги з низьким пріоритетом.

2. «Інтерактивні» процеси (I/O-bound) залишаються в чергах з високим пріоритетом.

3. Процес, який провів багато часу в черзі з низьким пріоритетом, «піднімається» у чергу з вищим пріоритетом («старіння»=aging).

08.3. Алгоритми планування: Multilevel Feedback Queue Scheduling



Приклад Multilevel Feedback Queue,
що складається з 3-х черг