① ⟶ multi-tasks → quantum?

→ Pilares:
- divisão justa da CPU p/ todos os processos em execução
- escalonador deve seguir a política de escalonamento p/ escolher um processo
- balancear a quantidade de processos p/ core
  - ↳ SMP: Um sistema com vários processadores com memória compartilhada sob o controle de um único SO.

1.2 def Processo: um grupo de threads que compartilham o mesmo TGID (thread group ID) ⊕ recursos necessários ⟷ não diferencia uma thread de um processo

Processo ↔ Thread

escalonador: escolhe uma thread

---

④ Fila (main Runqueue): cada CPU tem uma (cfs_rq) → CFS: justo; runqueue
↳ fair = justo
↳ contém as threads que estão prontas.

Componentes p/ Fila:
- locks: sincronização para escalonamento
  - ↳ raw_spinlock lock;
- ponteiros p/ as threads: → curr em execução, idle e → stop parada

---

⑤ rschdule(): é método usado para **invocar o escalonador**
↳ escolhe qual será o próxima a thread e coloca para executar

5.1
↳ Objetivo: 1º encontrar a thread a ser escalonada
2º associá-la a variável local next.
3º executa uma troca de contexto

obs: Se uma nova thread não é encontrada, a prev, se ela não for concluída suas tarefas (ainda resta tempo de execução), ela será escalonada novamente

→ Preemptivo → uma thread T_1 rodando, e se aparecer uma T_2 com uma prioridade maior; T_1 será preemptada e T_2 escalonada.

→ preempt_disable(): p/ a thread do schedule não será preemptada nas partes críticas ①

→ pega o lock da rq da CPU → apenas uma thread pode modificá-la ②

→ ③ prev
- I) acabou e não será preemptada? => remover da rq → dequeue_task(). → deactivate_task()
- II) c/ sinais (não bloqueados) pendentes? => def. estado como RUNNING => colocar na rq

→ ④ Existe alguma thread c/ o estado RUNNING na rq?
- ↳ N: idle_balance() => pega uma de outra CPU.
- ↳ S: pick_next_task() => pegar a nova thread conforme a prioridade
  - ↳ é a anterior? S/ troca de contexto => bota p/ executar
  - ↳ N: troca de contexto (context_switch()) { mapa da memória, regs, pilha

⑤ unlock rq ⑦ preempt_able()

5.2 Quando o escalonador é chamado?

① Atualização dos valores de uma thread
- ↳ scheduler_tick(): chamada regularmente pela interrupção de relógio.
- ↳ atualiza { clock da rq (?) → o que é? → Quantum?, CPU load, contadores

tag need_resched: avisa o kernel para invocar schedule

② A thread usa sleep()
- I) Thread cria uma fila de espera => adiciona ela mesma.

(II) vai ficar em um loop até uma certa condição acontecer

↳ wake_up() : sinal pendente

(III) chama schedule

(3) Uma thread que estava dormindo acorda

wake_up() try_to_wake_up() (ttwu) :

1. Coloca a thread que será acordada na r_q
2. Estado = RUNNING
3. Se ela tem mais prioridade que a thread em execução, a tag Need_Resched é setada → invoca schedule()

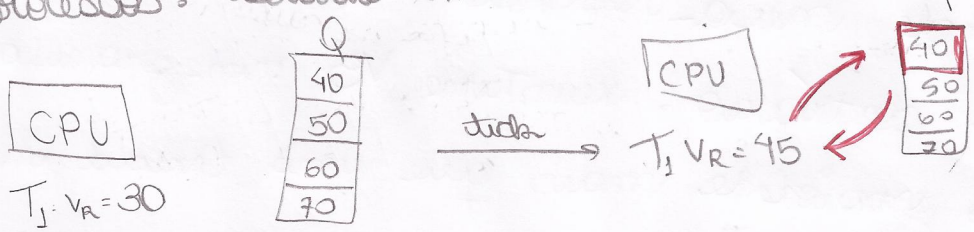
---

(7) CFS : Teoria => cada processo teria uma porcentagem da força da CPU => Impossível!

vruntime = incrementado de acordo com o tempo na CPU

↑ prioridade ↓ vruntime → Estado pronto

↳ novos processos: setado min_vruntime



TSL: granularidade mínima definida

↳ Qual o melhor valor?

Prioridade: { alta : T_R cresce devagar; baixa : T_{VR} cresce rápido

Interativo: ↑ prioridade

---

⑧ Estrutura de Dados

sched_entity: contadores de uma thread.

cfs: ponteiros p/ as threads em execução
raiz da árvore
ponteiro para o nodo mais a esquerda
min_vruntime
ponteiros prev e curr

Contador: (I) Atualiza os valores da árvore
(II) Verifica se a curr precisa ser preemptada

↳ delta: quanto aumentar do vruntime?
↳ chamado quando → executável/dormiu
sched_slice(): retorna a maior cadência (TSL) de tempo de execução da curr, depende da quantidade de processos prontos (runTime)
↳ grande: need_resched set
↳ caso contrário: runTime VS min_granularity
↳ maior se mais que uma thread na árvore

A comparação é feita com o elemento mais a esquerda
↳ vruntime ≥ 0 => curr executou tempo demais e ficou maior que a TSL => need_resched.