Task scheduling.

Whenever we have a set of tasks t. and to check if this set is schedulable, we need to check if it exceeds CPU. utilization or not.

$$t = (C_i, T_i, D_i)$$

C_i: CPU time that t_i will take to complete its execution.

T_i: the time interval after which t_i's second instance will be released.

D_i: the deadline before which that t_i should complete its execution.

$D_i = T_i$ is common assumption for real time task system

*high priority

What is Real time task RTOS

Def: Tasks that must complete within strict time to ensure system correctness. (deadline)

→ Time-critical (hard deadline).

Execute at regular intervals,

Execute time must be bounded and known.

high priority. React quickly to external events.

which means summing up all utilization of CPU per task Ti is still not max which is 100% of cpu capacity.

scheduling req:

1. Must check CPU utilization : $\boxed{\sum (Ci / Ti) \le 1}$.

2. Common algo: RMS, EDF (earliest deadline first).
(Rate Monotonic)

ex. ABS brake control - execute every 10ms.
汽車ABS剎車系統

Video decoding - process frame every 33ms.

Sensor data sampling - read every 100ms.

Game rendering - render every 16 ms.

Flight Control System - continuous monitoring.

low priority

vs. Batch tasks.

Def: Tasks can be collected and processed together, without strict timing constrains.

→ Non time-critical, Deferrable, Bulk processing, Background execution. Throughput-oriented.

ex. Database backup. bank statement.

Bulk email sending

log processing

(sigma)
for single processors that should exist a func $\sigma(t)$ which maps t to a set of tasks containing runnable and idle tasks $\sigma(t): t \to T \cup (T_{idle})$

for multiprocessors (setup with M CPUs)

$\sigma(t)$ should map to a matrix of task

$\sigma(t) := \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$ M=2, $\begin{cases} \text{Global scheduling} \\ \text{partition.} \end{cases}$

Global scheduling ↗ single shared ready queue for all CPUs.

partition: each CPU has its own task queue.

Dhall's Effect.

Even if $\sum \frac{C_i}{T_i} \le M$, tasks set may still be unschedulable on M processors!

ex. M processors, M+1 tasks

M heavy tasks (T, T, T) → utilization ≈ 1 each.

1 light task $(1, T, T)$ → " ≈ 0.

total ⇒ $M + 1/T$.

as $T \to \infty$: utilization → M (seems ok!)

Reality: M heavy tasks occupy all processors,

↳ light task can't execute → Miss ddl.

⇒ hard limit task: Task with utilization close to 1.0.

In this case, EDF, RM can't be used,
because they're not meant for hard tasks.

Takeway:

Total utilization ≤ M ≠ schedulable!

↓

CBS Constant Bandwidth Server. → Used in RTOS (real time tasks where they have strict deadline.
hard limit tasks.

- Reserve fixed CPU bandwidth for each task.
- Task allocation: $t_i = (Q_i, P_i)$ → guarantee.

  Budget (quota (max runtime),

  Budget depleted → task throttled.
  (over) (限流)

  If task wants more Q_i, { decrease priority. / wait for next period.

- Maintain dynamic deadline.
  - Execution history
  - Allocated bandwidth.
  - current budget.

The CBS algo ensures that when a task wake up, it will get its share of cpu After the alloted wait time.

Linux default scheduler for normal tasks.

CFS complete Fair Scheduler

↳ care about If all processes get the fair share of their resources, not specialized for RT (like CBS)

terminology:

→ more weight, more priority.

weight (nice value) -20 (highest) ~ +19 (lowest)

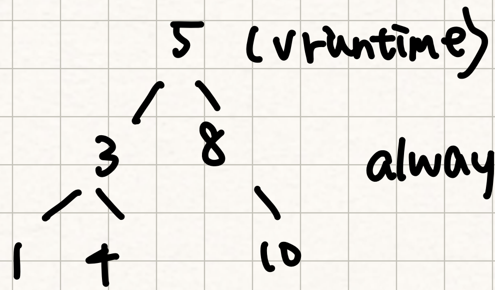
Timeslice: Dynamic interval, each task runs at least one. (not fixed)

virtual runtime (vruntime): = actual runtime × (weight of default / weight of task)

↳ ex. high priority (low nice value) = vruntime grows slowly.

vice versa.

Runqueue: Red-Block Tree.



always pick reftmost node (min runtime)

How CFS work?

1. Pick task with MIN (vruntime)
2. Run for calculated timeslice.
3. Update vruntime based on weight.
4. Re-insert into RBTree.
5. Repeat..

ex. Task A: weight = 1024 (normal)

B: weight = 2048 (high priority)

same actual runtime:

Task A vruntime = 100ms × 1024/1024 = 100.

B = 100ms × 1024/2048 = 50.

run more
(lower vruntime)

☆ Formula

Ideal runtime: sched_period × (task weight / total weight)

prevent too small timeslice.

Bottom line: CFS ensures cpu time <u>proportional</u> to weight radio.

成正比