O(1) Scheduler – 2.4 Big-oh

↳ Multi-level feedback queue scheduler

→ low time-complexity, so suitable for a lot of processes

→ could handle interactive and CPU-intensive processes.

Disadvantages: –

(1) A lot of heuristics are involved.

(2) Abrupt changes in the Time slices.

Completely Fair Scheduler

Target assigning of equal processing time to all the processes.

Suppose there are n processes.

$$ts \propto \frac{1}{n}$$

$$ts = \frac{k}{n}$$, k is a constant of proportionality, and here it is known as "target latency".

Process runtime r_i of process P_i

$ts - \Delta$ → Waiting state.

$ts \neq \Delta$

How much Time in running state

$$\frac{20}{n=1} \qquad \frac{20}{1}$$

2 ms in running state ⟶ Least amount of runtime.
30 ms in waiting state

Scheduled for execution

$n=5$ $\qquad \frac{20}{5} = 4$ ms ✓

If n becomes large, then the time slice will gradually become small.

No process will receive a time slice smaller than 1 ms.

CFS ⟶ $\max\left(1, \frac{k}{n}\right)$ in milliseconds

is not in practice completely fair

How is it implemented?

- Keep track of the run time
- Using vruntime variable in sched_entity structure within the Linux kernel

Process with minimum vruntime

— Using a height-balanced tree

— AVL Tree } → O(log n)

— Red-black Tree 2

Rebalancing happens ~~ove~~ much more often

What are the system calls related to scheduling?

nice numbers → CFS also takes care of nice numbers.

How?

vruntime = runtime * [nice_number + 21]

① nice(1)

Increments the nice value by the given number.

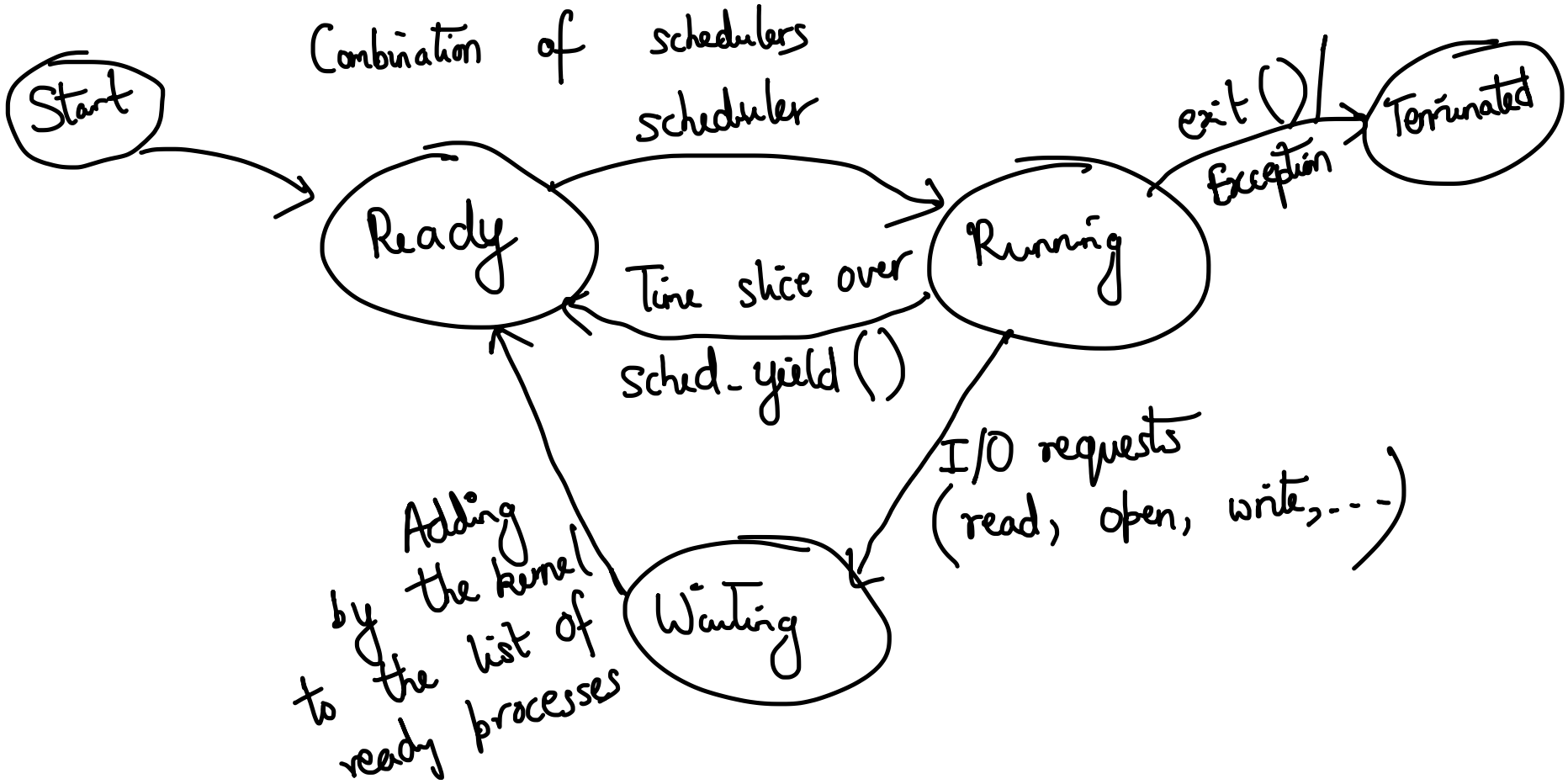
vruntime → unsigned long

② sched_yield();

Soft real-time jobs/processes

→ SCHED_FIFO, SCHED_RR

real-time processes


Combination of schedulers
scheduler
Start
Ready
Running
Terminated
exit()
Exception
Time slice over
sched_yield()
I/O requests
(read, open, write, ...)
Waiting
Adding by the kernel to the list of ready processes