CYBER-PHYSICAL SYSTEMS

- physical/engineered systems whose operations are monitored, coordinated and controlled by a reliable computing and communication core

  → automotive systems (autonomous driving, parking assistance, airbag controls)
  → avionics (flight navigation and control)
  → manufacturing systems (robotics, process controls)
  → medical systems (robotic surgery, devices)

- RTOS was need because :

① Proliferation of sensor devices.
these devices include embedded processors which ofc need an OS

② Systems being integrated as system of systems

- CPS need :

- Real time processing. The applications must process inputs and generate outputs correctly within a given deadline

- logical / functional + temporal correctness

- Applications : pacemaker, collision detection + avoidance systems

What should the deadline be?

① functionality?

→ collision avoidance in an automotive (milliseconds)
→ pacemaker (second)
→ robotic surgery (depends on target)

↑
longer timings are taken as design-time planning and not real-time requirements

② environment constraints?

→ available computing and communication resources
→ timing characteristics of sensors / actuators / operations

③ failure-mitigation strategies? ← design consideration

→ time to detect and recover from failures
→ consider timing overhead of strategies for detecting and recovering from failures
eg. replication of computing for redundancy.

THIS 'DEADLINE' IS THE WORST CASE TIME TAKEN FOR THE APPLICATION TO RESPOND

REAL TIME CPS / REAL TIME OS

- def: system whose correctness depends not only on logical / functional aspects, but also temporal aspects

- key performance measures:

① Timeliness / Predictability on timing constraints (deadlines)
② Significance of worst-case over average-case

how to model → RTOS Process

- def: Real Time Process: < R, C, D >
  - R: process release time
  - C: execution requirement
  - D: relative deadline

meaning: C time units in the CPU are needed between time [R, R+D)

for simplicity, we assume: 1 CPU burst (duration = C)
no I/O bursts

RECURRENT RTS

- most RTS are recurrent (pacemaker, collision avoidance)
- each instance of execution is a RTP <R, C, D>

PERIODIC RTP

- repeats periodically
- processes generated by a time triggered phenomena (eg. sensor sending data periodically)

- defined as <T, C, D>
  T : process period → release every T time units
  C : CPU time
  D : deadline

releaseds as R = 0, T, 2T ....

SPORADIC REAL TIME PROCESS

- repeats sporadically with a minimum gap between releases
- processes generated by an <u>event triggered phenomena</u> (eg. anti-lock braking function) ↳ induced by humans

- defined as <T, C, D>
  T : minimum release - separation time

short term scheduler / ready queue scheduler

RT CPU Scheduling

- classic algorithms like FCFS, SJF, RR fail as they don't prioritize deadlines

① Fixed Priority Scheduling

- all instances of the recurrent processes always have the same priority

VERY VERY POPULAR ←

A. Rate Monotonic Scheduler

- Priority is assigned based on process periods / minimum release separation time → $\underline{\underline{T}}$

shorter T ⇒ larger priority
Ties are broken arbitrarily

- gives a predictable schedule for high-priority periodic sporadic process

- ∵ RM ignores D, it can still miss a deadline

B. Deadline Monotonic (DM) Scheduler

- Assigns priorities based on process deadlines (D)

Shorter D ⇒ greater probability
Ties are broken arbitrarily

- ∵ since DM can't change priorities at the level of process instances, it can miss deadlines for certain process sets.

2. DYNAMIC PRIORITY SCHDULER
Earliest Deadline First Scheduler

- dynamic priority scheduler assigns priorities based on process instance deadlines

- instances with shorter dealines are given higher priority
- ties are broken arbitrarily

RM / DM	EDF
Simpler implementation (separate queue for each recurrent process)	Harder implementation (online sorting of queue based on instance deadlines)
Predictability for high priority process, even under overload	misbehavior during overload