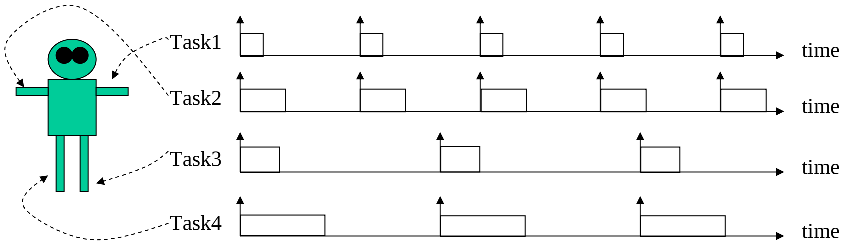
Overview of
Commonly used Approaches to
Real-Time Scheduling
- Chapter 4 -

Overview of Chapter 4 topics

- The nature of the "game"
- Overview of common approaches
  - Clock-Driven Approach
  - Weighted Round-Robin Approach
  - Priority-Driven Approach
    - EDF, LST, RM
- How to prove a scheduling is optimal
- Optimality in limited sense

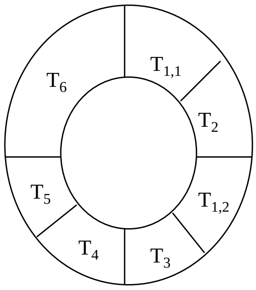
How to schedule?

- Four control-law tasks
  - Task 1 (left arm control): p1=30ms
  - Task 2 (right arm control): p2=30ms
  - Task 3 (left leg control): p3=50ms
  - Task 4 (right leg control): p4=50ms



The Clock Driven Approach

- It is also known as cyclical executive or timer driven approach.
- Scheduling decisions are made at specific time instants. These instants are chosen a priori before the system begins its execution.
- The timer kicks off the execution of a segment of code from a table

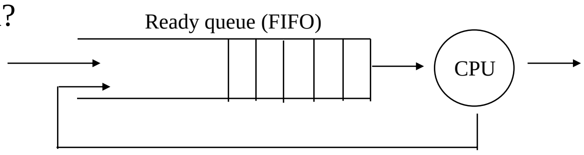


It is still in use and useful for
- small and simple embedded systems
- safety critical flight control systems

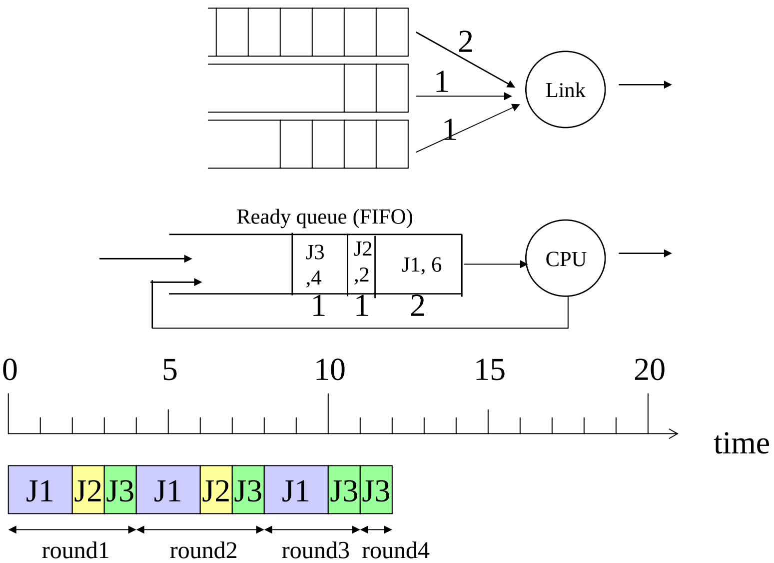
(slowly phased out by static priority scheduling)

Weighted Round Robin (WRR) Approach

- It is the old time token passing approach with a twist. When the token pass to you, you get a slice of CPU. If you have nothing to do or you finish early, the token is given to the next guy right away – scheduling message transmissions in ultrahigh-speed networks
- A time slice is typically in the order of tens of milliseconds.
- WRR let some tasks to hold the token longer (larger slice) than others
- Quiz: How does WRR or RR differ with Clock driven approach?



Weighted Round Robin (WRR) Approach



Priority Driven Approach

- This approach assigns priority to tasks. The highest priority task present in the ready queue gets the CPU and:
  - runs to completion or
  - is preempted by a higher priority job that has arrived.

- Quiz: Ok. Tasks have priorities, highest priority task gets the CPU. FIFO seems the ultimate "anti-priority" algorithm. Why is it classified as one of the priority scheduling schemes in scheduling theory?

- Answer: FIFO "assigns" higher priorities to the tasks arriving earlier. Theoretically, any algorithm that orders tasks in someway and never let CPU idle whenever there is a ready job is called a priority scheduling scheme (or ***work conserving*** scheme).

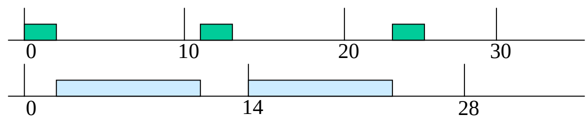
Priority Management

- Per task: (static priority) every job in the same task has the same priority. RM algorithm: the higher the rate, the higher is the priority of the task.
- Per Job: (dynamic priority) each job in a task can have different priority. e.g.,: EDF.
- Per time-tick: jobs priority will be reassigned after each tick, e.g., LST

- Can you tell me in the following diagram which 2 algorithms are used? And which is which.

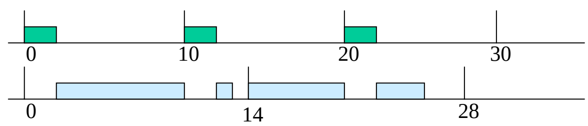
T1 = (10,2)

T1 = (14,9)



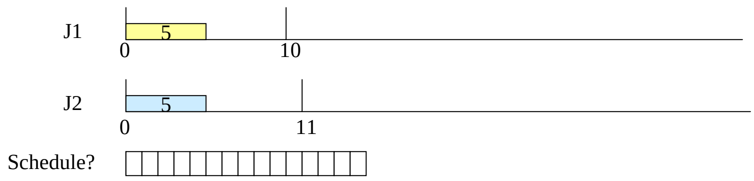
T1 = (10,2)

T1 = (14,9)

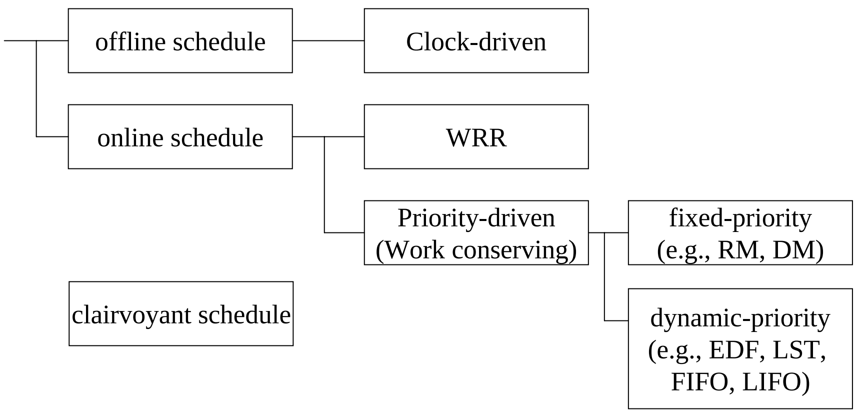


Least Slack Time (LST)

- Per time-tick: (also called dynamic priority): jobs priority will be reassigned after each tick, e.g., LST
  - Slack is the distance between the deadline and the job completion time, assuming that the job gets the processor
  - When a job is executing, its slack _______________
  - When a job is suspended, its slack ______________.
  - At some time t, the slacks of two jobs become equal. We break the tie and let J1 execute first.
  - After 1 clock tick, which task will execute next and why?
  - …
  - What is wrong with LST? __________________



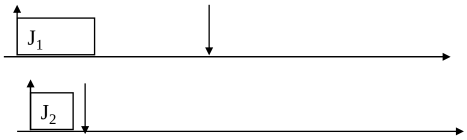
Classification of Scheduling Algorithm



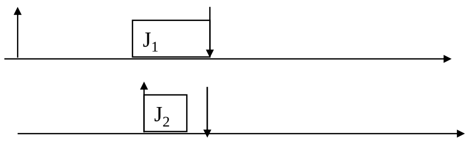
Is any known algorithm good? -“Scary” Results-

- When job arrivals are unpredictable, offline schedule does not have much a chance. Online schedulers perform poorly. Research showed that in worst case it can only schedule 0.25 of what could be scheduled if we would know the future.

- Consider the case of a non-preemptable job J_1 which becomes ready at t=0.



Suppose that we run J1 right away. J2 comes later and misses its deadline



Suppose we run J1 as late as possible. Guess what? J2 could have come late being unschedulable.

Life Goes On

- Fortunately, in real-time systems, you do not need to have a “crystal” ball to be clairvoyant most of time.

- The predominate hard real-time tasks are periodic. You are “clairvoyant” in this case.

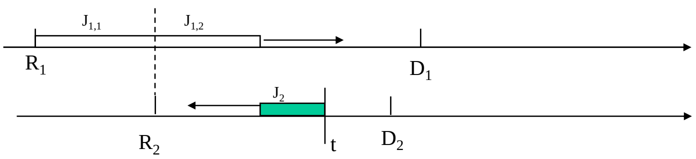
- Randomly arrival jobs mostly are soft real-time, so performance is statistically determined and there cannot be a “clairvoyant” adversary to generate jobs against you.

Optimality

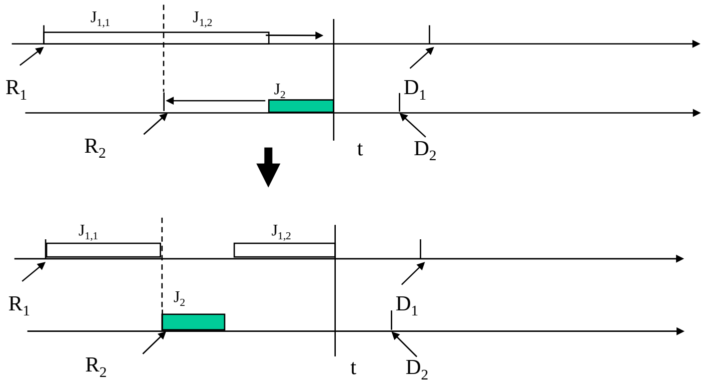
- **<u>The art of "me too"</u>** : A scheduling algorithm S is optimal under some giving condition, if any algorithm can schedule a set of tasks, so can S.
- Other algorithms can tie, but cannot beat S.
- So the key of proving the optimality is to demonstrate that you have a way to transform any (successful) schedule into a schedule produced by your algorithm.

The Swapping Trick - 1

- A key step (technique) in comparing schedules is the "swapping trick".
- ***How can we prove that EDF is optimal?*** Giving a schedule (that meets all the deadlines), we need to show that we can transform it into an EDF schedule.
- At R_2 both J_1 and J_2 are ready, we will let J_2 go first since $D_2 < D_1$. That is, we swap the execution of J_{12} with J_2



The Swapping Trick - 2

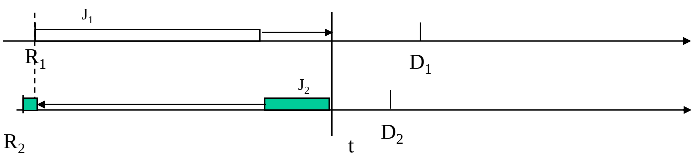


Swapping is legal if it observes given scheduling constraints. Swapping $J_{1,2}$ with J_2 is legal, because:

- Execute J_2 earlier at R_2 meets release time constraint and J_2 still meets deadline D_2.
- $R_2 + (J_{1,2} + J_2) = R_2 + (J_2 + J_{1,2}) = t < D_2 < D_1$

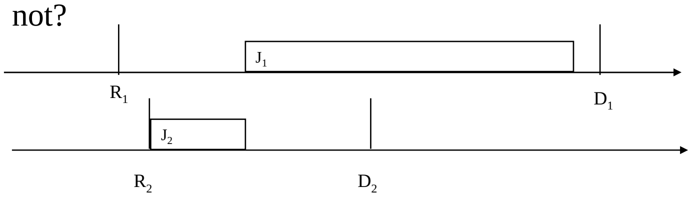
Any pair, not a particular pair

- We have given an example of legally swapping two tasks. But to claim EDF is optimal, we need to show that we can swap any two jobs not in deadline order. The swapping trick can apply to any two jobs not in deadline order.
  - We can always execute job J_2 earlier at its release time. This cannot cause J_2 missing its deadline
  - After swapping, J_1's finishing time becomes J_2's finishing time t. Since $t < D_2 < D_1$, swapping cannot cause J_1 to miss its deadline D_1 either.



Work Conserving/Nonconserving Schedules

- Quiz: Is this schedule produced by EDF? Why and why not?

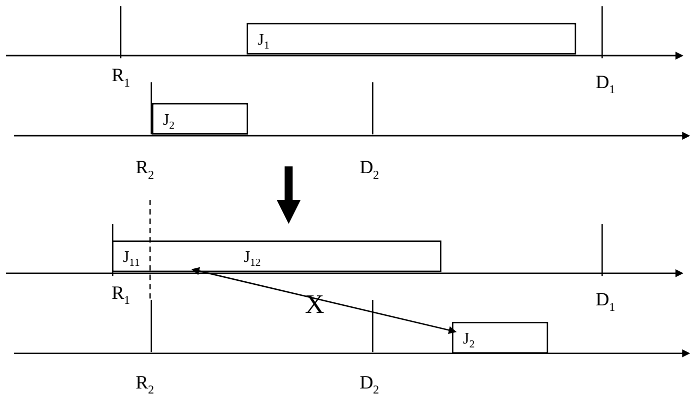


- Quiz: Is this schedule produced by any priority scheduler at all? Why and why not.

- Priority schedule is said to be work conserving, meaning that if there is a job ready to execute, we can't let the processor idle.

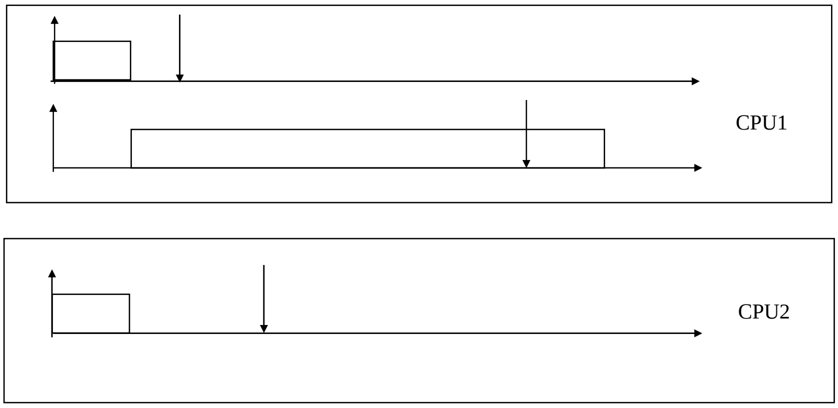
Non-preemptive EDF
(Can't Preempt, Can't Swap)

Answer: EDF, as with any priority scheduling algorithm, CANNOT let processor idle. Hence J_1 must execute first. Unfortunately, when J_2 is ready, it cannot preempt J_1 since jobs are not preemptable

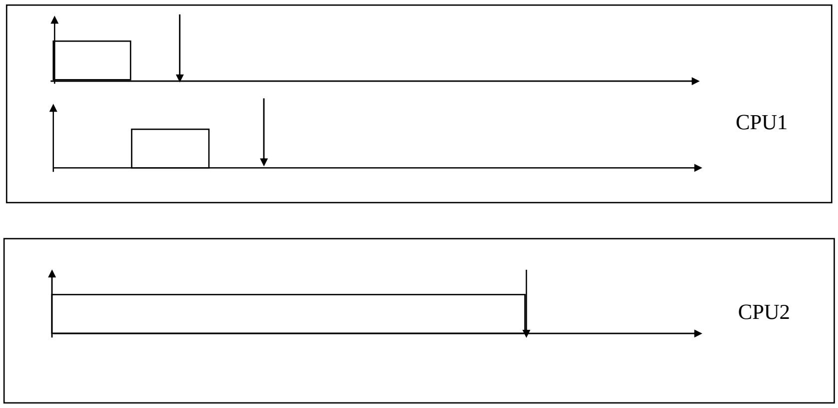


Quiz: Is non-preemptive EDF optimal? Why and why not?

EDF on more than one processor



Feasible schedule on more than one processor



Is EDF optimal with multiprocessor systems?

Summary of EDF Optimality

- Optimality of EDF: EDF is optimal when preemption is allowed, only a single processor is used, and jobs are independent

- To prove the optimality of a scheduling algorithm S under a giving set of constraints (rules), we must show that under these rules any successful schedule can be transformed into a S schedule.

- Trick of the trade is the swapping operation that
  - Observes the constraints
  - And tries to transform any successful schedule into a S schedule by swapping.