Clock-Driven Scheduling

Notations and Assumptions

- Clock-driven approach applicable on DETERMINSTIC (with some exceptions) systems.
- Assumption: A restricted periodic task model.
  - n periodic tasks in the systems.
  - Parameters of all tasks known a priori: each job in T_i is released p_i units of time after the previous.
  - Each job $J_{i,k}$ is ready for execution at $r_{i,k}$.

Notations and Assumptions

- A periodic task *Ti* is referred to by (ϕ_i, p_i, e_i, D_i)

Where:

ϕ_i is the phase,

p_i is the period,

e_i is the execution time,

D_i is the relative deadline.

- Example: (1, 10, 3, 6)

Notations and Assumptions

- Example: (1, 10, 3, 6)
  - First job released and ready at time 1,
  - Must be completed by time 7,
  - Maximum execution time 3,
  - Second job ready at 11, must be completed by 17.
  - Utilization of task = 0.3.

Notations and Assumptions

- Default value of Phase is zero,
- Default value of Relative Deadline = Period.
- Elements with default values are omitted.

- Aperiodic Jobs released at unexpected jobs.
- Tasks scheduled on One processor.

Static Time-Driven Scheduler

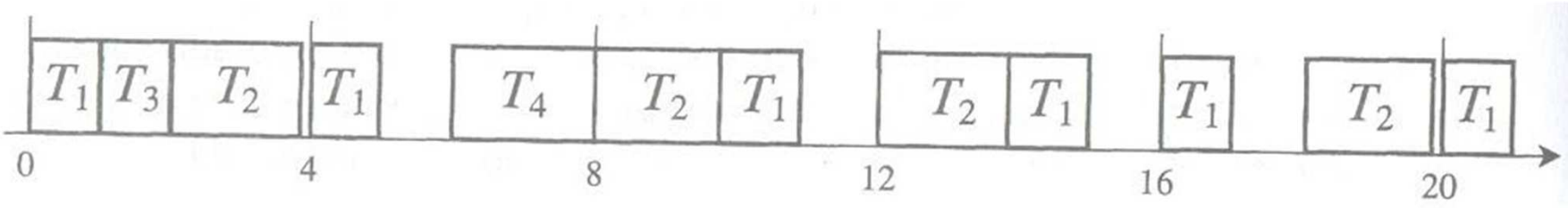
- A queue is maintained for aperiodic jobs.
- Aperiodic jobs are ordered in this queue in a suitable manner.
- When parameters of Hard real-time jobs are known, a *static schedule* is constructed offline.
- The scheduling algorithm (can be complex) is chosen to satisfy some criteria.

Static Time-Driven Scheduler

- Four independent tasks:
  - T_1 = (4, 1),
  - T_2 = (5, 1.8),
  - T_3 = (20, 1),
  - T_4 = (20, 2).
- Hyperperiod = 20.
- Schedule: Replication of segments of length 20.

Static Time-Driven Scheduler

- Periodic static schedule: Cyclic schedule.

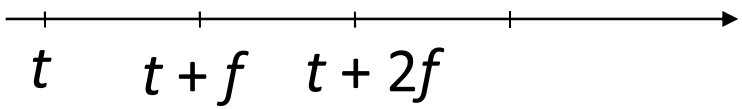


General Structure of Cyclic Schedules

- Use a schedule with a certain structure.
- If the structure is “good” according to some criteria, then the schedule is “good”.

Frames and Major Cycles

- Scheduling decisions made periodically.
- Intervals between 2 consecutive decisions are Frames. f is the frame size.
- No preemption during a frame.



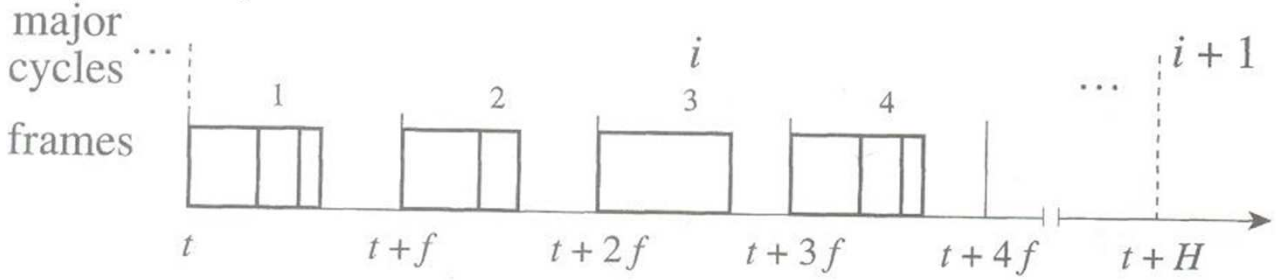
- Monitoring enforcing actions at the beginning of each frame.

Frame size constraints

- No preemption ➔ f larger than execution time e_i of every task T_i.
- Length of the cyclic schedule as short as possible ➔ f divides H ⇔ f divides the period p_i of at least one task T_i.
- Each hyperperiod contains an integer number, F, of frames).
- *Major cycle*: Hyperperiod that begins at the beginning of the $(kF + 1)_{k = 0, 1, \ldots}$ frame.

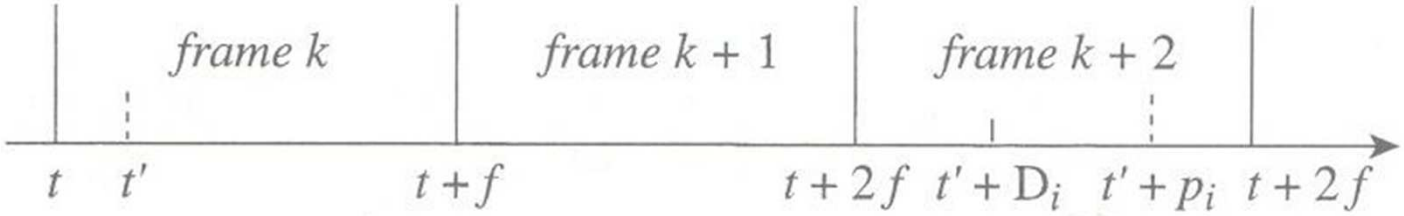
Frame size constraints

- Major Cycle:



Frame size constraints

- In the fig. below, *t'* is release time of a job.



- We must have for all *i*:

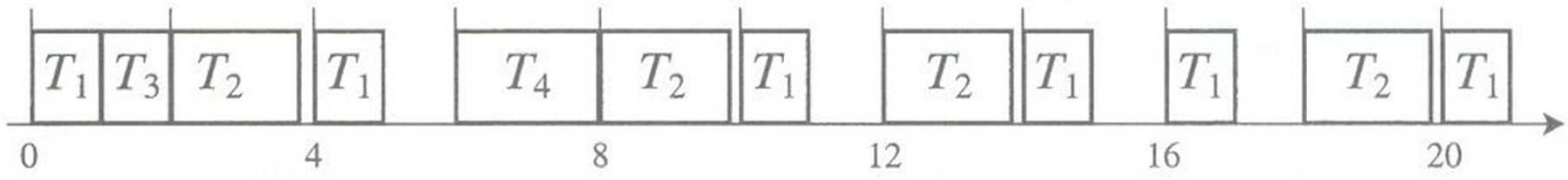
$$t + 2f \le t' + D_i$$

$$\Leftrightarrow \quad 2f - (t' - t) \le D_i$$

$$\Leftrightarrow \quad 2f - gcd(p_i, f) \le D_i$$

Frame size constraints: Example

- Four independent tasks:
  - $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$, $T_4 = (20, 2)$.
- $f \geq \max(e_i)$ ➔ $f \geq 2$,
- f divides H ➔ $f = 2, 4, 5, 10$ or 20.
- $2f - gcd(p_i, f) \leq D_i$, $f = 2$.

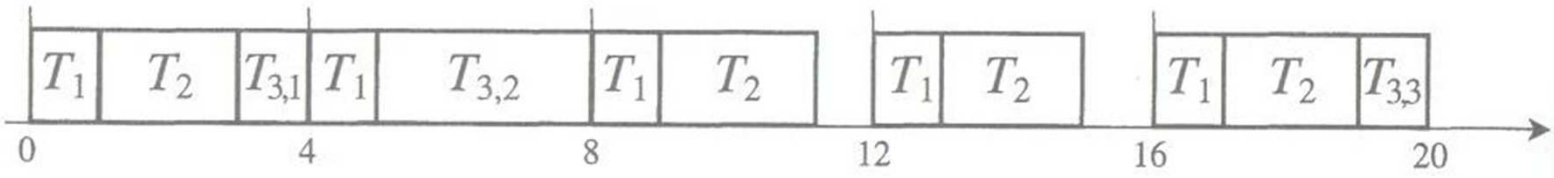


Frame size constraints: Example

- Three tasks:
  - T_1 = (15, 1, 14), T_2 = (20, 2, 26), T_3 = (22, 3)
- $f \geq 3$,
- f = 3, 4, 5, 10, 11, 15, 20 or 22.
- f = 3, 4 or 5.

Job slices

- Sometimes, all frame size constraints cannot be met simultaneously.
- **T** = {(4, 1), (5, 2, 7), (20, 5)}, $f \geq 5$ and $f \leq 4$!!!
- Solution: Partitioning each job in a task with a large execution time into slices.
- (20, 5) = (20, 1)-(20, 3)-(20, 1) and $f = 4$.



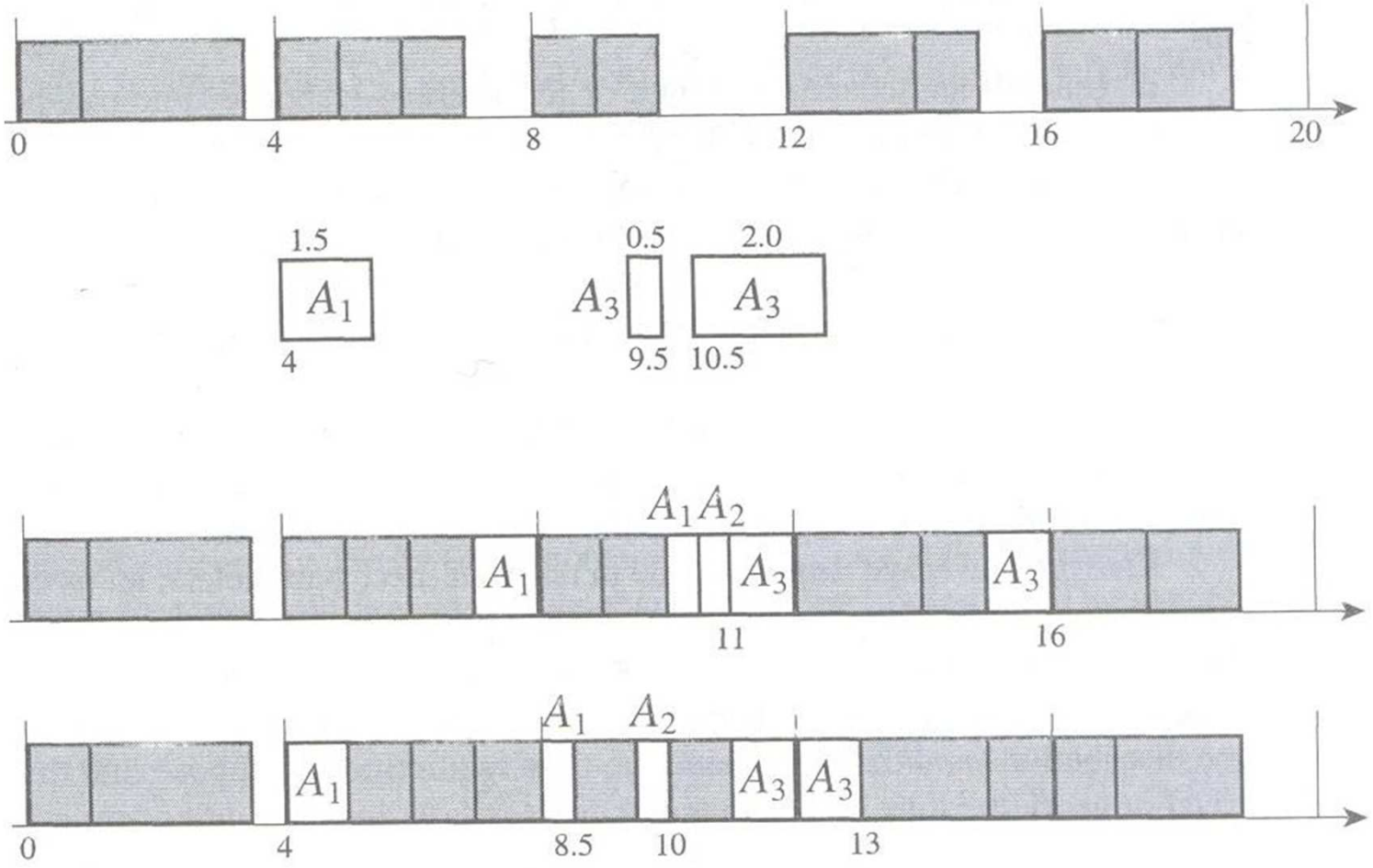
Improving the Average Response Time of Aperiodic Jobs

- Aperiodic jobs, so far, scheduled after periodic jobs.
- The sooner an aperiodic job completes, the more responsive the system is ➔ Minimizing the avergae response time of aperiodic jobs.

Slack Stealing

- Slack stealing: Executing , when possible, the aperiodic jobs ahead of the periodic jobs.
- Every periodic job slice to be scheduled in a frame that ends no later than its deadline.
  - x_k: time allocated to all slices scheduled in frame k.
  - $f - x_k$: the slack available at the beginning of the frame.
  - Aperiodic jobs, if any, can be executed for this amount of time.

Slack Stealing: Example



Average Response Time

- Average response time no greater than some value ➔ Need to estimated this average.
- Accurate estimation can be found by simulation and/or measurement. Cons: time consuming , requires that system is designed/built.
- Queuing theory technique can be applied. Statistical behavior of aperiodic jobs must be known: Average rate of arrival, mean and mean square of execution times.

Average Response Time

- n_a: aperiodic tasks,
- λ_i: average rate of arrival of jobs in task *i*. unit: jobs per unit of time.
- E[β_i] and E[β_i^2]: mean and mean square values of execution times in task *i*.
- u_i: average utilization (fraction of processor time required by jobs) of task *i*.
- $u_i = \lambda_i$ E[β_i]. U_A = sum of u_i : Average utilization of aperiodic tasks.

Average Response Time

- U: total utilization of periodic tasks.
- $1 - U$: time available for the execution of aperiodic tasks. We consider the case where $U_A < 1 - U$.

$$W = \sum_{i=1}^{n_a} \frac{\lambda_i E[\beta_i]}{\lambda(1-U)} + \frac{W_0}{(1-U)^2\left[1-U_A/(1-U)\right]}$$

$$W_0 = \sum_{i=1}^{n_a} \frac{\lambda_i E\left[\beta_i^{\,2}\right]}{2}$$

Scheduling Sporadic Jobs

- Temporal parameters of sporadic jobs are unknown a priori ➔ No guarantee a priori that all sporadic jobs complete in time.

Acceptance Test

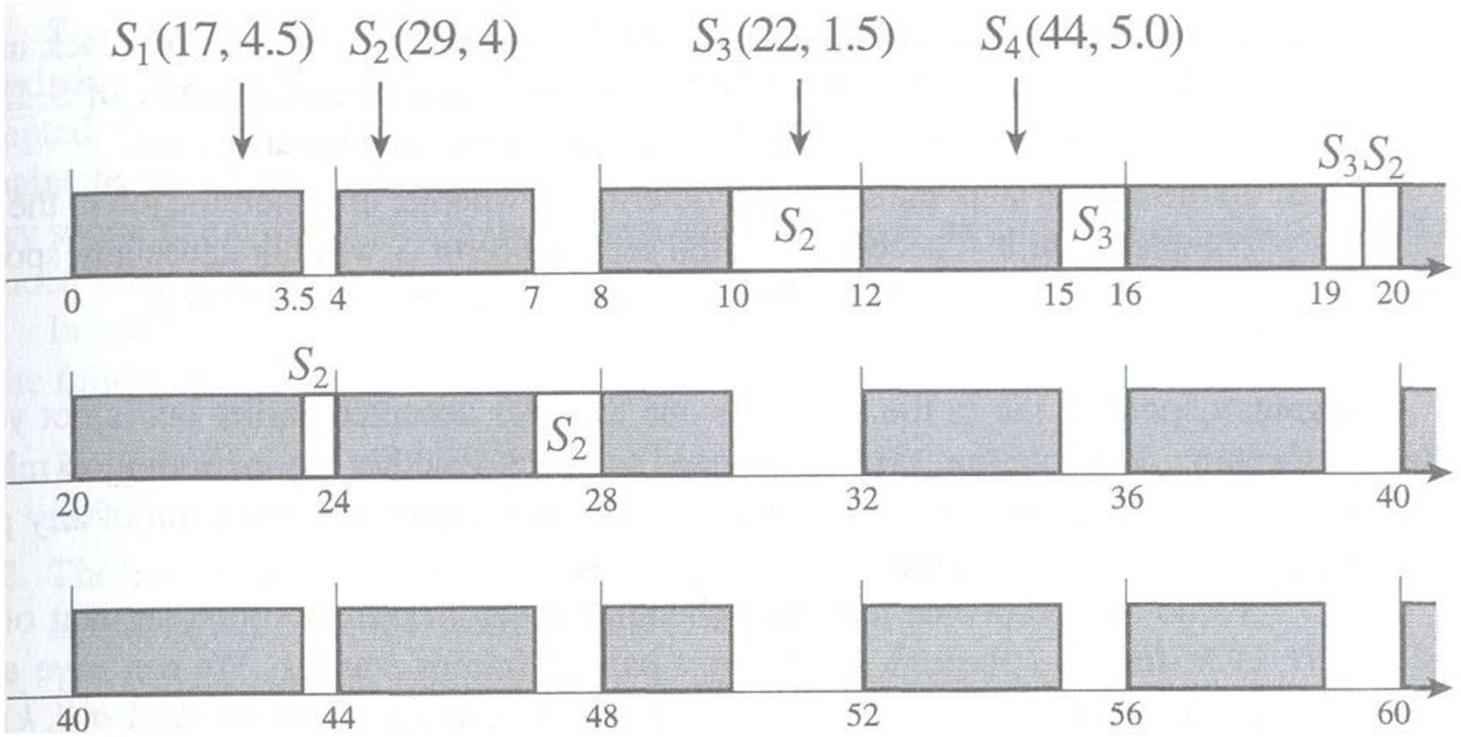
- During an Acceptance Test, the scheduler checks if the released Sporadic Job can be feasibly scheduled with ALL* jobs in the system.
- *: Periodic and other sporadic jobs previously scheduled.

EDF Scheduling of the Accepted Jobs

- The max execution time known when job is released.
- Acceptance tests done at the beginning of the frame following the instant where the sporadic job is released.

EDF Scheduling of the Accepted Jobs

- S_1(deadline, execution time) and S_4 rejected.



Pros and Cons of clock-driven scheduling

- Pros:
  - Simple,
  - Easy to validate and to test.
- Cons:
  - Difficult to modify and maintain,
  - Release times of all jobs must be fixed,
  - Not suitable for system containing both hard and soft applications.