ELC 4438: Embedded System Design

Liang Dong

Electrical and Computer Engineering
Baylor University

Reference Model of Embedded Systems

- A reference model focuses on

  the timing properties and

  resource requirements of system components and

  the way the operating system allocates the available system resources among them.

Reference Model of Embedded Systems

- According to the reference model, a system is characterized by:
- A workload model that describes the applications supported by the system
- A resource model that describes the system resources available to the application
- Algorithms that define how the application system uses the resources at all times.

Processors and Resources

- System resources: processors and resources

- Processors – active resources P_n
  - Examples: CPUs, transmission lines, disks
- Resources – passive resources R_m
  - Examples: memory, sequence number, database locks
  - Examples: computation job shares data with other computations, data guarded by semaphores; communication ACK sequence number

- The elements of a system can be modeled as processors or resources depending on the use of the model.

Processors and Resources

- A resource is reusable – it is not consumed during use.
- A reusable resource may have one or more units, each unit is used in a mutually exclusive manner.
- A resource is plentiful if no job is ever prevented from execution by the lack of this resource. e.g. memory can be a plentiful resource.
- Omit the resources that are plentiful.

Temporal Parameters of Real-Time Workload

- The workload on processors consists of jobs, each of which is a unit of work to be allocated processor time and other resources.
- The number of tasks is known a priori before the system begins execution.
- Job J_i is characterized by its temporal parameters, functional parameters, resource parameters, and interconnection parameters.
- Assume that the temporal parameters of hard real-time jobs/tasks are known.

Temporal Parameters

- Release time r_i, absolute deadline d_i, relative deadline D_i, feasible interval of job J_i $(r_i, d_i]$
- The range of r_i is known but not the exact value. Release time jitter in r_i $[r_i^-, r_i^+]$.
- Negligible jitter → fixed release time
- Sporadic jobs / aperiodic jobs are released at random time instants in response to external events. Release time of sporadic jobs are random variable. To model, use A(x), the arrival time distribution (or interarrival time distribution).

Temporal Parameters

- Execution time e_i of job J_i. The amount of time required to complete the execution of J_i when it executes alone and has all the resources it requires.
- Execution time depends on complexity of the job and the processor speed, not on scheduling.
- The min and max execution time of J_i are assumed known of every hard real-time job $[e_i^-, e_i^+]$
- Execution time e_i usually means max execution time.

Deterministic Approach

- We commonly use deterministic approach first, because the hard real-time portion of the system is often small.

- Reclaim the time and resources allocated but not used by hard real-time jobs and make them available to the rest soft real-time jobs and nonreal-time jobs.

Periodic Task Model

- The periodic task model is a deterministic workload model.
- A periodic task, T_i, is computation or data transmission that is executed repeatedly at regular time intervals to provide a function on a continuing basis.
- A periodic task is a sequence of jobs.

Parameters of Periodic Model

- Period p_i of T_i is minimum length of time intervals between release times.

- Execution time is the maximum execution time of all jobs in T_i

- The accuracy of the periodic task model decreases with increasing jitter in release times and variation in execution times.

Parameters of Periodic Model

- The phase of T_i, ϕ_i , is the release time $r_{i,1}$ of the first job $J_{i,1}$ in task T_i
- Tasks in phase
- Hyperperiod of the periodic tasks
- Utilization of the task T_i

Aperiodic and Sporadic Tasks

- Aperiodic and sporadic tasks model the workload generated in response to unexpected events.

- Each aperiodic or sporadic task is a stream of aperiodic or sporadic jobs.

- The interarrival times in each task vary drastically.

Aperiodic and Sporadic Tasks

- The jobs in each task model the work done by the system in response to events of the same type.

- The jobs in each aperiodic task have the same statistical behavior and the same timing requirement, $A(x)$.

Aperiodic and Sporadic Tasks

- The execution times of jobs in each aperiodic or sporadic task are identically distributed random variables, B(x).

- With distributions A(x) and B(x), the system is stationary.

Aperiodic vs. Sporadic Tasks

- An aperiodic task has jobs that have either soft deadlines or no deadline.

- A sporadic task has jobs that are released at random time instants and have hard deadlines.

Precedence Constraints and Data Dependency

- Data and control dependencies among jobs may constrain the order in which they can execute.

- Jobs have precedence constraints if they are constrained to execute in some order.

- Jobs are independent if they can execute in any order.

Precedence Constraints and Data Dependency

- $J_i < J_k$, J_i is a predecessor of job j_k, j_k is a successor of J_i.
- Immediate predecessor / immediate successor
- J_i and J_k are independent if neither $J_i < J_k$ nor $J_k < J_i$
- A job with predecessors is ready for execution when the time is at or after its release time and all of its predecessors are completed.

Precedence Graph and Task Graph

- Precedence Graph represents the precedence constraints among jobs in a set **J**.
- Each vertex represents a job in **J**.
- A directed edge from vertex J_i to J_k when the job J_i is an immediate predecessor of J_k.
- A task graph is an extended precedence graph. A task graph may contain different types of edges representing different types of dependencies.

Data Dependency

- Data dependency can not be captured by a precedence graph.
- In a task graph, data dependencies among jobs are represented explicitly by data-dependency edges among jobs.
- There is a data-dependency edge from J_i to J_k if the job J_k consumes data generated by J_i or the job J_i sends messages to J_k.
- Edge parameters: e.g. Volume of data from J_i to J_k.

Other Types of Dependencies

- Temporal dependency
- AND/OR precedence constraints
- Conditional branches
- Exclusive Access to Resources
- Pipeline relationship of periodic schedules

Functional Parameters

- Several functional parameters affect scheduling and resource access-control decisions:

- Preemptivity
- Criticality
- Optional Executions
- Laxity Type

Preemptivity of Jobs

- Preemption – scheduler suspends the execution of a less urgent job and gives the processor to a more urgent one. Afterwards, returns the processor to the less urgent job to resume execution.

- A job is preemptive if its execution can be suspended at any time, and later on can be resumed from the point of suspension.

- A job is nonpreemptive if it must be executed from start to completion without interruption.

Criticality of Jobs

- The criticality of a job indicates how important the job is with respect to others.

- During an overload when it is not possible to schedule all the jobs to meet their deadlines, the less critical jobs are sacrificed so that the more critical jobs can meet their deadlines.

Optional Executions and Laxity Type

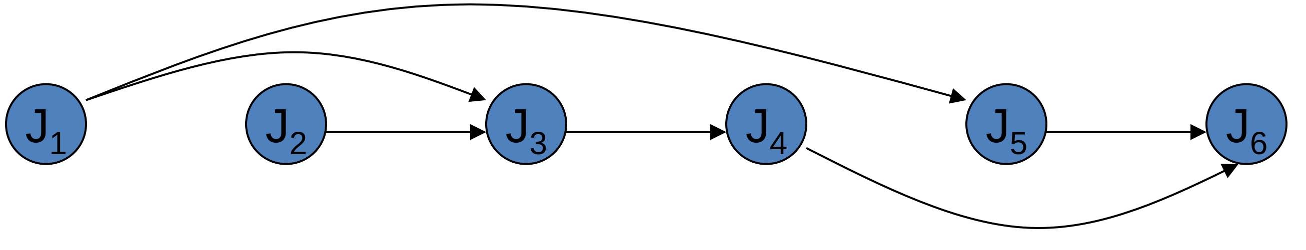
- If an optional job completes late or is not executed at all, the system still can function but the performance may degrade.
- Laxity – lack of strictness
- The laxity type of a job indicates whether its timing constraints are soft or hard.
- The laxity function is given by the usefulness function of its tardiness.

Real-Time Scheduling

Prelude: Maximal Parallelism

- Question: Given a precedence graph, how many processors should be used to execute it?
- Example – Sequential program:

```
a := x + y;      /* Job J1 */
b := z + 1;      /* Job J2 */
c := a - b;      /* Job J3 */
w := c + 1;      /* Job J4 */
d := a + e;      /* Job J5 */
w := w * d;      /* Job J6 */
```

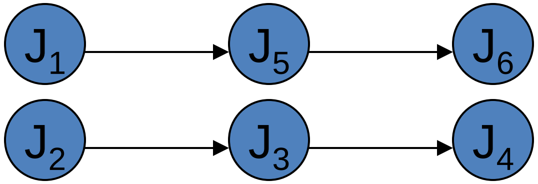


Maximal Parallelism

- Assign processors 1 and 2:

P_1:

P_2:



- Synchronize J_1 with J_3, and J_4 with J_6.
- Answer: "maximally parallel"
  – The maximum number of processors that can be used efficiently is equal to the cardinality of the largest set of nodes such that there is no dependency between any two nodes in the set.