Lecture 5

Task classes

Tasks can be classified in two ways: By the predictability of their arrival and by the consequences of their not being executed on time.

Periodic, aperiodic and sporadic:

Periodic tasks consist of an infinite sequence of identical activities, called *instances* or *jobs*, that are regularly activated at a constant rate.
Aperiodic tasks also consist of an infinite sequence of identical jobs (or instances); however, their activations are not regularly interleaved. An aperiodic task where consecutive jobs are separated by a minimum inter-arrival time is called a *sporadic task*.

Critical and non-critical tasks:
Critical tasks are those whose timely execution is critical; if deadlines are missed catastrophes occur.
Non-critical tasks: are non-critical to application.

Definitions:
Arrival time *ai* is the time at which a task becomes ready for execution; it is also referred as *request time* or *release time* and indicated by *ri*;
Computation time *Ci or Exi* is the time necessary to the processor for executing the task without interruption;
Absolute Deadline *di* is the time before which a task should be completed to avoid damage to the system;
Relative Deadline *Di* is the difference between the absolute deadline and the request time: $Di = di - ri$;
Start time *si* is the time at which a task starts its execution;
Finishing time *fi* is the time at which a task finishes its execution;
Response time *Ri* is the difference between the finishing time and the request time: $Ri = fi - ri$;
Criticality is a parameter related to the consequences of missing the deadline (typically, it can be hard, firm, or soft);
Value *vi* represents the relative importance of the task with respect to the other tasks in the system;

Lateness Li: $Li = fi - di$ represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative;
Tardiness or *Exceeding time* Ei: $Ei = max(0, Li)$ is the time a task stays active after its deadline;
Laxity or *Slack time* Xi: $Xi = di - ai - Ci$ is the maximum time a task can be delayed on its activation to complete within its deadline.

PRECEDENCE CONSTRAINTS

In certain applications, computational activities cannot be executed in arbitrary order but have to respect some precedence relations defined at the design stage. Such precedence relations are usually described through a directed acyclic graph G, where tasks are represented by nodes and precedence relations by arrows. A precedence graph G induces a partial order on the task set.

The notation $Ja \prec Jb$ specifies that task Ja is a *predecessor* of task Jb, meaning that G contains a directed path from node Ja to node Jb.

The notation $Ja \rightarrow Jb$ specifies that task Ja is an *immediate predecessor* of Jb, meaning that G contains an arc directed from node Ja to node Jb.

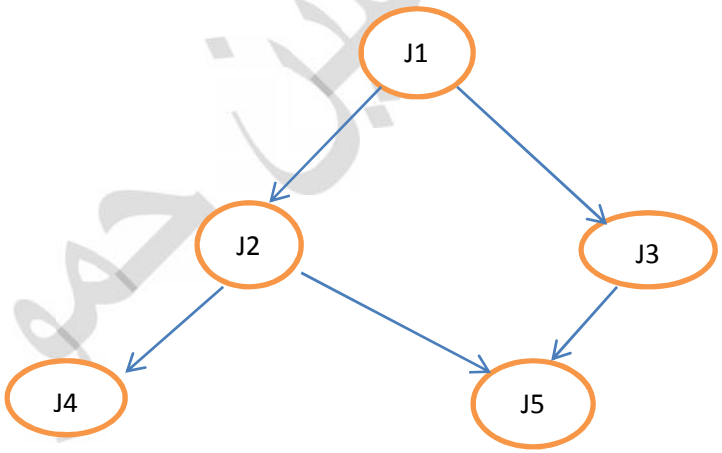

Figure 2 Precedence relations among five tasks

Figure 2 illustrates a directed acyclic graph that describes the precedence constraints among five tasks. From the graph structure we observe that task $J1$ is the only one that can start executing since it does not have predecessors. Tasks with no predecessors are called *beginning tasks*. As $J1$ is completed, either $J2$ or $J3$ can start. Task $J4$ can start only when $J2$ is

completed, whereas $J5$ must wait for the completion of $J2$ and $J3$. Tasks with no successors, as $J4$ and $J5$, are called *ending tasks*.