Event-Driven Scheduling
(closely following Jane Liu’s Book)
Principles

Admission: Assign priorities to Jobs
At events, jobs are scheduled according to their “priorities”

Important properties:
• decisions, which job to execute next at events (not time instants) such as releases and completions of jobs
• a (timer) interrupt is an (implementation of a) special event
• never leaves a resource idle intentionally (“greedy”)
• scheduling on line,
  admission on line or off line
• scheduling must be simple (otherwise not possible on line)
Restrictions Given Up

some “restrictive” assumptions of time-driven systems are given up:

• fixed inter-release times  $\rightarrow$ minimum inter-release times

• fixed number of rt tasks in systems  
  $\rightarrow$ real-time and non real-time, number can vary

• a priori fairly well known parameters  
  $\rightarrow$ tasks come and go, overloading, ...
Priority Assignment Following “Criticality”

The more critical a task the higher its priority

T1: (2,0.9)  T2: (5,2.3)

T2 more critical than T1

T1 misses deadline in Job 1 and 2/3, unnecessarily ...

---

Real-Time Systems, 2008  
Event-Driven Systems, 4  
Hermann Härtig, TU-Dresden
Important Variants

• **Static vs dynamic allocation to processors**
  • static: jobs are assigned to processors once and stay there
  • dynamic: one queue served by all processors (jobs “migrate”)

• **static vs. dynamic priorities**
  • static: jobs do not change their priorities (unless new tasks arrive)
  • dynamic: priorities are recomputed frequently

  e.g., FIFO is dynamic priority scheduling

• **preemptive or non preemptive**
  • some tasks
  • all tasks
Preemptive vs. Non-Preemptive Scheduling, Example

2 processors,

Tasks: Notation used below: \( J_i, e_i \)

- release time of \( J_5 \) is 4, all others 0; (!)

Static priorities, assigned such that:
- \( i < k \Rightarrow \text{Prio}(J_i) \text{ higher than } \text{Prio}(J_k) \)

Tasks can “migrate”

Precedence graph:

\[
\begin{align*}
J_1,3 & \quad \rightarrow \\
J_2,1 & \quad \rightarrow \\
J_5,2 & \quad \rightarrow \\
J_7,4 & \quad \rightarrow \\
J_3,2 & \quad \rightarrow \\
J_4,2 & \\
J_6,4 & \\
J_8,1 &
\end{align*}
\]
Example, executions

P1
0 4 8 12
J1 J4 J7 J6

P2
J2 J3 J5 J8

P1
0 4 8 12
J1 J4 J5 J6

P2
J2 J3 J7 J8

preemptive

J1,3
J2,1
J5,2
J4,2

non preemptive

J3,2
J6,4
J8,1
Modified Example: release time of J5 = 0

P1
0 4 8 12
J1 J5 J6

P2
J2 J3 J4 J7 J8

P2
J2 J3 J4 J7 J8

non preemptive

J1,3
J2,1
J5,2
J7,4

J3,2
J6,4
J8,1

J4,2
Which is better?

No general answer known!

If jobs have same release time:
preemptive is better (or equal) in a multiprocessor system if
cost for preemption is ignored
more precise: “makespan” is better
(makespan = response time of job that completes last)

how much better?
Coffman and Garey:
2 processors:
makespan(non-preemptive) <= 4/3 *
makespan(preemptive)
**Effective Release Times and Deadlines**

“Inconsistencies” due to precedence relations

- a release time given for a job may be later than that of its predecessor
- a deadline may be earlier than of its successor time
From Now: use effective ...

Effective Release Time:
- of a job without predecessors: the given release time
- of a job with predecessors: \[ \text{max} \left( \text{given release time, effective release times of all predecessors} \right) \]

Effective Deadline:
- of a job without successor: the given deadline
- of a job with successor: \[ \text{min} \left( \text{given deadline, effective deadlines of all successors} \right) \]
Earliest Deadline First

Assign priorities at run time ...

“the earlier the deadline the higher the priority”

**Theorem:**
- One processor.
- Jobs preemptable.
- Jobs do not contend for passive resources.
- Jobs have arbitrary deadlines, release times.

Then: EDF is “optimal”, i.e.
- if there is a feasible schedule,
- there is also one with EDF
EDF Optimality

Proof: (informal)
assume a feasible, non EDF schedule
systematically transform it to an EDF schedule (3 steps)

1. 

2. 

3. 

r_k

Non EDF

Ji Jk Ji Jk

Jk Ji Jk

Jk Ji Jk

Jk Jk Ji

Jk Jk Ji

Jk Jk Ji

Jk Jk Ji

d_k d_i
Earliest Deadline First, priority assignment:

fixed per job, dynamic at task level:
the nearer the absolute deadline of a job at release time
the higher the priority
T1: (2,0.9)   T2: (5,2.3)
Latest Release Time (LRT)

Rationale:
no need to complete rt-jobs before deadline
use time für other activities

Idea:
Backwards Scheduling
Run as late as possible
Use latest possible release times as „priority“

optimal (analog EDF-Definiton of Optimality)
Example (Precedence Graph):

\[ J_{1,3} (0,6) \rightarrow J_{2,2} (5,8) \]

\[ J_{3,2} (2,7) \]
Least Slack Time First / Minimum Laxity First

Slack Time = Laxity:
(time to deadline
– remaining time required to reach deadline)

also optimal (analog EDF definition)
Least Slack Time First

dynamic per job, dynamic at task level:
slack time: \( d - x - t \)
\( x \) remaining execution time of a job
\( d \) absolute deadline
\( t \) current time

two versions:
• strict:
  slacks are computed at all times (prohibitively slow)
• non-strict:
  slacks computed only at events (release and completion)
scheduler checks slacks of all ready jobs and reorders queue
Non-Strict LST Example

T1: (2,0.75)  T2:(5,1.5)  T3: (5.1,1.5)

0  2  4  8  12  16  20  24  28

\[ t=0 \]
all Jobs released
\[
T1,J1: 1.25 \quad T2,J1: 3.5 \quad T3,J1: 3.6
\]
d.h. T2,J1 higher priority than T3,J1

\[ t=2 \]
T1,J2 released
\[
T1,J2: 1.25 \quad T2,J1: 2.75 \quad T3,J1: 1.6
\]
d.h. T2,J1 lower priority than T3,J1

\[ t=2.75 \]
T1,J2 completed
\[
T1,J2: \quad T2,J1: 2 \quad T3,J1: 0.85
\]
EDF and Non-Preemptivity

Job: (release time, execution time, deadline)
J1: (0,3,10)  J2: (2,6,14)  J3: (4,4,12)

EDF is not optimal if jobs are not preemptable.
EDF and Multiple Processors

Job: (release time, execution time, deadline)
J1: (0, 1, 2)   J2: (0, 1, 2)   J3: (0, 5, 5)

dealine missed

EDF is not optimal for Multiprocessors.

easy for time driven schedulers
Scheduling Anomaly

<table>
<thead>
<tr>
<th>J1</th>
<th>Release</th>
<th>Deadline</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td>0</td>
<td>10</td>
<td>[2,6]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>varies</td>
</tr>
<tr>
<td>J3</td>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>J4</td>
<td>0</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Increasing priorities:
\[ i < k \Rightarrow \text{Prio}(J_i) \text{ higher than Prio}(J_k) \]

2 processors, preemptable but not migratable

Intuitive approach:
Check for worst case (a) and best case (b) execution times
and be confident ...
Scheduling Anomaly, cont

a

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Task J1] 4 8 12</td>
<td>![Task J2] 4 12</td>
</tr>
<tr>
<td>![Task J3] 8 16</td>
<td>![Task J4] 8 16</td>
</tr>
</tbody>
</table>

b

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Task J1] 4</td>
<td>![Task J2] 4</td>
</tr>
<tr>
<td>![Task J3] 4 12</td>
<td>![Task J4] 8 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Task J1] 4</td>
<td>![Task J2] 4</td>
</tr>
<tr>
<td>![Task J3] 4 12</td>
<td>![Task J4] 12 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Task J1] 4</td>
<td>![Task J2] 4</td>
</tr>
<tr>
<td>![Task J3] 4 12</td>
<td>![Task J4] 12 20</td>
</tr>
</tbody>
</table>

Real-Time Systems, 2008  
Event-Driven Systems, 22  
Hermann Härtig, TU-Dresden
Scheduling Anomaly on One Processor

Job: (release time, execution time, deadline)

J1: (0,3-4,10)  J2: (2,6,14)  J3: (4,4,12)

Not preemptable

release time job 3  deadline missed

E1=3

E1=4
Predictable Execution

Informal definition:

Given a set of periodic tasks with known minimal and maximal execution times and a scheduling algorithm.

A schedule produced by the scheduler when the execution time of each job has its maximum (minimum) value is called a maximum (minimum) schedule.

An execution is called predictable, if for each actual schedule the start and completion times for each job are bound by those of the minimum and maximal schedules.
Predictable Execution

The execution of every job in a set of independent, preemptable jobs with fixed release times is predictable when scheduled in a priority driven manner on one processor.
Validation Algorithms

... determine whether all jobs meet their deadlines
correct or not
accurate or not
• overly pessimistic
• overly optimistic
Assumptions for Next Set of Algorithms

Periodic set of tasks with these properties:
• Tasks are independent
• one processor
• no aperiodic or sporadic tasks
• preemptable, context switch is negligibly small
• period = minimum inter-release times (not fixed)

Since tasks are independent, tasks can be added (if admitted) and deleted at any time without causing deadline misses.
Priority Assignment

- **fixed priority:**
  fixed for task (and jobs) relativ to other tasks

- **dynamic priority:**
  priority of tasks changes at release and completion times in relation to other tasks
  - fixed per job
  - dynamic per job
Rate Monotonic Scheduling

fixed priority:
the shorter the period the higher the priority
(rate: inverse of period)

element: T1: (4,1)  T2: (5,2)  T3: (20,5)
Deadline Monotonic Scheduling

fixed priority:
the shorter the relative deadline the higher the priority

eample: \((\phi, P, e, D)\)

\(T_1: (50, 50, 25, 100)\) \(T_2: (0, 62.5, 10, 20)\) \(T_3: (0, 125, 25, 50)\)

Conclusion (no proof): DM better than RM if \(D\) arbitrary
(More) Comparison Criteria

- Optimality
- Validation

- Schedulable Utilization (SU) of an algorithm: a scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if $\varepsilon/p \leq SU$.
  
  SU: the higher the better.
  Dynamic priority schedulers better than fixed priority.

- Predictability in the presence of overload: in fixed priority systems it is possible to predict which tasks are affected due to overruns.
Priority-Driven Scheduling of Periodic Tasks

To do:
- admission (required before new tasks are admitted)
- priority assignment (off line / on line)
- selection of next task (on line)

restrictions (whether they apply or not)
- dependencies (precedence, sharing)
- multiple processors
- aperiodic, sporadic

achievable resource utilization: $U = \frac{\text{e}}{p}$
EDF and Multiple Processors

Job: (release time, execution time, deadline)
J1: (0, 1, 2)    J2: (0, 1, 2)    J3: (0, 5, 5)

EDF is not optimal for Multiprocessors.

easy for time driven schedulers
Another Multiprocessor Example

$m$ processors, $m+1$ tasks

$m > 0, \ m*2^\square < 1, \ \square \text{small}$

$T_i, \ i=1..m:$ \hspace{1cm} Period 1, \hspace{0.5cm} execution time: $2^\square$

$T_{m+1}:$ \hspace{1cm} Period $1+\square$ \hspace{0.5cm} execution time: 1

scheduler: priority (edf or shortest period first)

allocation: dynamic

discuss!

Pathological cases, mostly dynamic performs better
very hard to analyze for worst case
EDF and Overload, examples

T1: (2, 1)  T2: (5, 3)  \( U=1.1 \)

T1 misses

T1: (2, 0.8)  T2: (5, 3.5)  \( U=1.1 \)

T1 und T2 miss

No easy way to determine which jobs miss deadline ...
EDF and Overload, one more example

T1: (2, 0.8) T2: (5, 4.0) U=1.2

J2,1 continues to execute after deadline and ... causes J1,3 to miss the deadline
Utilization: RM ./. EDF

T1: (2,1)  T2: (5,2.5)

U = 1

EDF

RM

T2 misses deadline

RM not optimal in general
Optimality of Fixed Priority Schedulers

T: periodic tasks, independent, preemptable, one proc.

**Deadline Monotonic:**
relative deadlines $\leq$ periods, in phase
if there is any feasible fixed priority schedule for T, then Deadline Monotonic is feasible as well

**Rate Monotonic:**
relative deadlines = periods
simply periodic, i.e.
for all pairs of tasks $i,j$: if $P_i \leq P_j$ holds $P_j = n \times P_i$
RM is schedulable iff $U \leq 1$ (cmp. EDF)
Some Schedulable Utilization(SU) Results

indep. tasks, preemptable, relative deadline=period, one processor

N Number of Tasks

EDF: SU = 1

\[ RMS: \quad SU = n \left(2^{1/n} - 1\right) \quad n \cdot J : \quad \ln(2) \]

RMS (simply periodical, D \( \square \) P): SU = 1
Schedulibility Test for Fixed(!) Priority

(case where jobs must complete before end of period)

Critical Instant Analysis / Time Demand Analysis:

critical instant for task $Ti$:
one of the jobs of $Ti$ is released at same time with a job in every higher priority task ...

It is sufficient to check a schedule for the critical instant for the longest envolved period
(Fixed Prio) Schedulibility and Blocking

Ti may have to wait for
non-preemptable, lower priority task

\( b_i: \)
longest non-preemptable portion of all lower prio. Jobs

Schedulability for all tasks Ti with fixed priority scheduler x
\( S_{U_x}(i): \)
Scheduled Utilisation for scheduling method x with i tasks:

\[
U_i = \frac{e_1}{p_1} + \frac{e_2}{p_2} \ldots \frac{e_i}{p_i}
\]

\[
U_i + \frac{b_i}{p_i} \leq S_{U_x}(i)
\]
Non Negligible Context Switch Time

For Job level fixed priority schedulers ... :
i.e. each job preempts at most one other job
2 context switches:
release (when it preempts other)
completion

include CS overhead in wcet:
\[ WCET_i := WCET_i_{original} + 2CS \]
(Fixed Prio and) Limited Priority Levels

Required: Mapping of

• Scheduling-Priorities: 1 ... n to
• Operating System Priorities: E, 2, ... µ

Jobs running with same OS-Prio but different Sched-Prio use: FIFO, Round Robin, ...

Schedulibility loss?

• Notation: as grid on Scheduling Priorities
• Example:
  10 scheduling priorities, 3 OS priorities
  possible mapping: 3 =3, 8 = 8, 10 = 10
  interpretation:
  0,1,2,3 mapped to 1, 4,5,6,7,8 to 2, 9,10 to 3

How is Schedulibility Test affected?
(Fixed Prio and) Limited Priority Levels

Mappings:

- **uniformly distributed:**
  \[ k = \frac{n}{m} \]
  Scheduling Priority X mapped to \( |X/m| \ast k \)

- **constant ratio:**
  keep \((\frac{c_{i-1}}{c_i} + 1)\frac{c_i}{c_i}\) as equal as possible
Schedulibility Loss

Rate Monotonic, large $n$ ...

$g = \min(\frac{n-1}{b} +1) / b$

$SU_{RM} = \ln(2g) + 1 - g$

Relative schedulibility (rs): relation to $\ln(2)$

Example:

$n = 100000$, $m = 256$

$rs = 0.9986$

$\Rightarrow$ 256 priorities is it!