Online Scheduling of Aperiodic Tasks

Contact

m.nasri@tudelft.nl

(appointment by email)
Online scheduling of aperiodic tasks

Buttazzo’s book, chapter 3

Disclaimer: Some slides have been taken from Giorgio Buttazzo’s website:
http://retis.sssup.it/~giorgio/rts-MECS.html
Summary of the previous quiz

How many deadline misses in the FIFO schedule?

Task 1

\[ C_1 = 5 \]

Task 2

\[ C_2 = 10 \]

Task 3

\[ C_3 = 1 \]

Total misses: 2
Summary of the previous quiz

What is bad about FIFO?

Task 1
- $C_1 = 5$
- 5
- 17
- 19

Task 2
- $C_2 = 10$
- 4
- 15
- 20

Task 3
- $C_3 = 1$
- 7
- 10
- 20

miss
FIFO: the judgement day

• Disadvantages of FIFO
  • Creates a non-preemptive schedule
  • Ignores task’s deadline and priority
  • Response times heavily depend on the arrival times
  • Low schedulability

• Advantages of FIFO
  • Fast
  • Simple implementation (easily implementable on hardware or software using a queue)
  • Low runtime overhead
  • Wide applicability in network systems (almost everywhere you have a queue)

*Schedulability*: the ability of the scheduling algorithm to generate feasible schedules for various task sets
Summary of the previous quiz

Is the task set feasible?

Yes! Here is one feasible schedule for it:

Task 1

Task 2

Task 3

\[ C_1 = 2 \]

\[ C_2 = 10 \]

\[ C_3 = 1 \]
Summary of the previous quiz

What scheduling policy do you think it is?

Yes! Here is one feasible schedule for it:

Earliest deadline first (EDF)

Task 1
- Start: 5
- End: 17
- Deadline: 17
- Completion time: 2

Task 2
- Start: 4
- End: 15
- Deadline: 15
- Completion time: 10

Task 3
- Start: 7
- End: 10
- Deadline: 10
- Completion time: 1

\[ C_1 = 2 \]
\[ C_2 = 10 \]
\[ C_3 = 1 \]
Summary of the previous quiz

Is it the only possible feasible schedule?

No! There can be more:

Yes! Here is one feasible schedule for it:

Task 1

\[ C_1 = 2 \]

Task 2

\[ C_2 = 10 \]

Task 3

\[ C_3 = 1 \]

\( C_1 = 2 \)

\( C_2 = 10 \)

\( C_3 = 1 \)
Summary of the previous quiz

In a *feasible schedule*, which task(s) might be executing at time 14? *(Think carefully)*

Only Task 2

If at time 14, any other task is executing, that schedule will not be feasible!
Agenda

• Scheduling complexity

• Online scheduling policies and their properties
Feasibility vs. schedulability

Space of all task sets

Infeasible task sets

Task sets schedulable by algorithm A

Task sets schedulable by algorithm B
General scheduling problem

Given a set $\tau$ of $n$ tasks, a set $P$ of $p$ processors, and a set $R$ of $r$ resources, find an assignment of $P$ and $R$ to $\tau$ that produces a feasible schedule under a set of constraints.
Complexity

• In 1975, Garey and Johnson showed that the general scheduling problem is **NP hard**.

In practice, it means that the time for finding a feasible schedule **grows exponentially with the number of tasks**.

Fortunately, polynomial time algorithms can be found under particular conditions.
Why do we care about complexity?

• Let’s consider an application with $n = 30$ tasks on a processor in which the elementary step takes $1 \mu s$ (one microsecond = $10^{-6}$ s).

• Consider 3 algorithms with the following complexity:

  $A_1$: $O(n)$
  - $30 \mu s$

  $A_2$: $O(n^8)$
  - 182 hours

  $A_3$: $O(8^n)$
  - 40,000 billion years

If you are not familiar with Big O notation to describe algorithm’s complexity, read wiki.
Textbooks:

“Complexity” (e.g., the big O) is a very important factor for choosing the right algorithm.

Yes! Complexity matters! A lot! It MATTERS! MMMAAATTERRRSS!

I agree, but what else?

Hmmm, If you insist, you can consider memory too: “Memory Complexity”!

As a real-time and embedded system engineer, what else should you consider?
Textbooks:

“Complexity” (e.g., the big O) is a very important factor for choosing the right algorithm.
Things you need to know about “complexity”

• Different implementations can have different runtime but similar complexity!

Algorithm A (input: n)
{
    a = now();
    b = a + 10;
    If(b > ready_queue.head()->deadline)
        Return ready_queue.head();
    Return IDLE_TASK;
}

Algorithm B (input: n)
{
    Return ready_queue.head();
}

They are both O(1), but Algorithm A is much slower than B
Things you need to know about “complexity”

- Different implementations can have **different runtime** but similar complexity!

```plaintext
Algorithm A (input: n)
{
    ----- -----
    ----- -----
    b = a * 3.14;
    ----- -----
}

Algorithm B (input: n)
{
    ----- -----
    ----- -----
    b = a + 3000;
    ----- -----
}
```

- “addition” is much faster in most microcontrollers than multiplication
- Integer operations are usually much faster than floating-point operations
Things you need to know about “complexity”

• Different implementations can have different runtime but similar complexity!

• Runtime of an algorithm depends not only on how many instructions you have to execute, but also on how the hardware platform executes it!

Some microcontrollers are very slow in floating-point operations

Microcontrollers support a limited set of instructions
Example: Arduino does not support “the most significant bit” instruction
Agenda

• Scheduling complexity

• Online scheduling policies and their properties
Simplifying assumptions

• Single processor
• Fully preemptive tasks
• No precedence constraints
• No shared resource or resource constraints
• No self-suspension
• Homogeneous task sets

(all tasks have the same type of activation, e.g., all are periodic, or all are aperiodic)
Properties of scheduling algorithms

- Preemptive v.s. non-preemptive
- Work-conserving v.s. non-work-conserving
- Static v.s. dynamic
- Offline v.s. online
- Optimal v.s. non-optimal (e.g., heuristic)
Work-conserving v.s. non-work-conserving

• Work-conserving
  Such algorithm **does not** leave the processor **idle** as long as there is a ready task in the system (a task is in the ready queue).

• Non-work conserving
  Such algorithm **may** leave the processor **idle** even if there is a ready task in the system.
Work-conserving v.s. non-work-conserving

Task 1

Task 2

Task 3

$C_1 = 1$

$C_2 = 10$

$C_3 = 4$

The processor is left idle in $[3, 5)$ even though Tasks 2 and 3 are in the ready queue.
Static v.s. dynamic

• Static
  
  scheduling decisions are taken based on fixed parameters, statically assigned to tasks before activation.

• Dynamic
  
  scheduling decisions are taken based on parameters that can change with time.

  Example:
  • remaining execution time
  • absolute deadline
Offline v.s. online

• Offline
  all scheduling decisions are taken before task activation: the schedule is stored in a table (table-driven scheduling).

• Online
  scheduling decisions are taken at run time on the set of active tasks.
Optimal v.s. heuristic

• Optimal
  They generate a schedule that minimizes a cost function, defined based on an optimality criterion.

• Non-optimal (heuristic)
  They generate a schedule according to a heuristic function that tries to satisfy an optimality criterion, but there is no guarantee of success.
Optimality criteria

Examples

• **Feasibility**: Find a feasible schedule if there exists one.

• Minimize the **maximum lateness** (also known as *makespan*)

• Minimize the **number of deadline miss**

• Assign a value to each task, then maximize the **cumulative value** of the feasible tasks
Common definitions for optimality criteria

Average response time:
\[ \bar{t}_r = \frac{1}{n} \sum_{i=1}^{n} (f_i - a_i) \]

Total completion time:
\[ t_c = \max_i (f_i) - \min_i (a_i) \]

Weighted sum of completion times:
\[ t_w = \sum_{i=1}^{n} w_i f_i \]

Maximum lateness:
\[ L_{max} = \max_i (f_i - d_i) \]

Maximum number of late tasks:
\[ N_{late} = \sum_{i=1}^{n} miss(f_i) \]

where
\[ miss(f_i) = \begin{cases} 
0 & \text{if } f_i \leq d_i \\
1 & \text{otherwise}
\end{cases} \]

Reference: Giorgio’s book, chapter 2, page 40
Let’s see how some of the scheduling algorithms work
Shortest-job first (SJF)

- Kind-of **Static** (if $C_i$ is a constant parameter)
- It can be used **online** or **offline**
- Can be **preemptive** or **non-preemptive**
- It **minimizes** the **average response time**

Note: Wikipedia says that the most common version of SFJ is “non-preemptive”. The following proof about average response time is also for non-preemptive tasks.
Optimality of SJF

After checking the book of Buttazzo, I noticed that the proof is not there.

Here are the assumptions for which the proof and hence the claim hold:
• SJF is non-preemptive (according to Wikipedia)

The story of the proof (since it does not exist in the book):

\( \sigma \) is a given schedule that is NOT compliant with SJF → Hence, it should have at least two tasks \textbf{Task L} and \textbf{Task S} that are scheduled with an order that is different from what SJF would have scheduled them. (Note: since SJF could have scheduled them differently, executing Task S before Task L must have been possible, namely, both tasks must have been released before the start time of Task L in \( \sigma \)).

Now, we create \( \sigma' \) by swapping these two tasks in the schedule and we show that \( \sigma' \) (that is more similar to an SJF schedule) has a smaller average response time
Optimality of SJF

To prove optimality of SJF, we need to prove that

\[ \bar{R}(\sigma') \leq \bar{R}(\sigma) \]

**Note:** This proof only holds for non-preemptive tasks.
Optimality of SJF

\[ R(\sigma) = \frac{1}{n} \sum_{i=1}^{n} (f_i - a_i) \]

Average response time

\[ R(\sigma') = \frac{1}{n} \sum_{i=1}^{n} (f'_i - a_i) \]

\( f'_S < f_L \)

\( f'_L = f_S \)

\[ f'_S + f'_L < f_L + f_S \]

\[ R(\sigma') = \frac{1}{n} \sum_{i=1}^{n} (f'_i - a_i) \leq \frac{1}{n} \sum_{i=1}^{n} (f_i - a_i) = R(\sigma) \]
Optimality of SJF

Proof by induction

Highlight: from any given schedule $\sigma$ we can make one that is more similar to SJF (by swapping two jobs that are not scheduled according to SJF)

\[
\sigma \rightarrow \sigma' \rightarrow \sigma'' \rightarrow \ldots \rightarrow \sigma^*
\]

\[
\bar{R}(\sigma) \geq \bar{R}(\sigma') \geq \bar{R}(\sigma'') \ldots \geq \bar{R}(\sigma^*)
\]

\[
\sigma^* = \sigma_{SJF}
\]

$\bar{R}(\sigma_{SJF})$ is the minimum response time achievable by any algorithm.
Is SJF suited for Real-Time?
Is SJF suited for Real-Time?

- It is not optimal in the sense of feasibility

\[ A \neq SJF \text{ feasible} \]

\[ SJF \text{ not feasible} \]
Fixed-priority scheduling

• Each task has a priority $P_i$, typically $P_i \in [0, 255]$
  ($P_i < P_j$ means Task $\tau_i$ has a higher priority than task $\tau_j$)

• The task with the highest priority is selected for execution.
• Tasks with the same priority are served FIFO order

$P_1 < P_2 < P_3$

Task 1

Task 2

Task 3

$C_1 = 1$

$C_2 = 8$

$C_3 = 4$
Fixed-priority scheduling

- Problem: starvation

  low priority tasks may experience long delays due to the preemption of high priority tasks.
Round robin

• The ready queue is served with FIFO, but ...
• Each task $\tau_i$ cannot execute for more than $Q$ time units ($Q = \text{time quantum}$).
• When $Q$ expires, $\tau_i$ is put back in the queue.
Round robin

- $n =$ number of task in the ready queue
- Round robin creates “rounds” that are as long as $n \cdot Q$
- Assume $Q = 1$

![Diagram showing task execution and CPU usage with Round 1, 2, and 3, and the corresponding CPU cycles for each task]
Round robin

Homework:

Derive an equation that provides an **upper bound** for the **worst-case response time** of the tasks scheduled by the round robin algorithm (the one we explained before)
Multi-level scheduling

- High priority tasks: PRIORITY
- Medium priority tasks: RR
- Low priority tasks: FCFS
Step 1: open www.kahoot.it in your browser (phone or laptop)
Step 2: enter the pin code and then a nickname
Quiz wrap-up

• What are the priorities used by the FP algorithm?

1: \( P_2 < P_1 < P_3 \)

2: \( P_2 < P_3 < P_1 \)

3: \( P_1 < P_2 < P_3 \)

4: \( P_3 < P_2 < P_1 \)

(P\(_i\) < P\(_j\) means Task \( i \) has a higher priority than task \( j \))
Quiz wrap-up

• What should be the priorities of FP so that it generates a FIFO schedule?

1: $P_2 < P_1 < P_3$

2: $P_2 < P_3 < P_1$

3: $P_1 < P_2 < P_3$

4: $P_3 < P_2 < P_1$

($P_i < P_j$ means Task $i$ has a higher priority than task $j$)
Quiz wrap-up

• Which scheduling algorithm is used?

1: FIFO
3: EDF
2: FP with priorities: $P_1 < P_2 < P_3$
4: preemptive shortest job first
Quiz wrap-up

• Which algorithm has a shorter maximum lateness?

1: FP
P1<P2<P3

Task 1

\[ C_1 = 5 \]

Task 2

\[ C_2 = 10 \]

Task 3

\[ C_3 = 1 \]

2: preemptive shortest job first

Task 1

\[ C_1 = 5 \]

Task 2

\[ C_2 = 10 \]

Task 3

\[ C_3 = 1 \]
Summary

• Complexity matters, but it is not the only thing that matters

• FIFO is fast and simple to implement, but bad at guaranteeing deadlines

• Non-preemptive SJF minimizes the average response time of the tasks

• Fixed-priority scheduling may starve low priority tasks

• Round robin has a large number of preemptions
Disclaimer

• The course slides came from Koen Langendoen who compiled materials found on the web and obtained from friendly colleagues together.

• Thanks to
  • Koen Langendoen
  • Akos Ledeczi
  • Gerd Gross
  • Giorgio Buttazzo
  • Reinder Bril
  • Zonghua Gu
  • and many others