ECE3849
D-Term 2021

Real Time Embedded Systems

Module 1 Part 2

Module 1 Part 2 Overview

- Interrupts with Preemption Summary
- Rate-Monotonic Scheduling Theory
  - RMS Graphical Model
  - Response-Time Analysis
  - Processor Utilization
  - Utilization Upper Bound

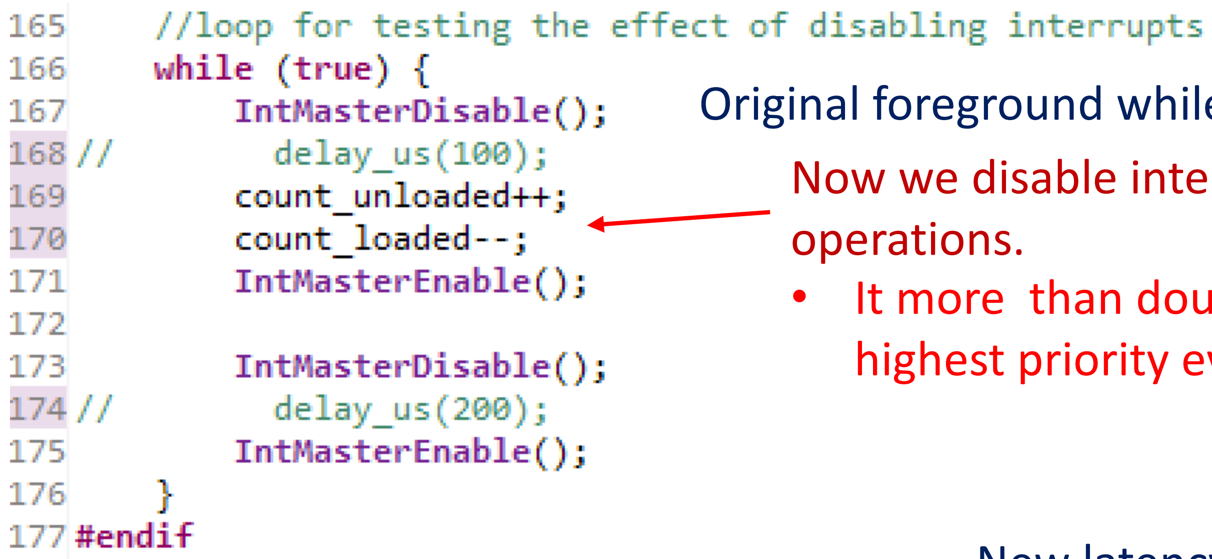
Preemptive Stategy Summary

- Canonical System assumptions / rules
  1. All events are periodic.
  2. The relative deadline = period.
  3. The events are not phase aligned and can happen at any time relative to each other.
- Preemptive strategy with interrupts.
  - Events with the shortest period are prioritized the highest.
  - Higher priority tasks preempt/interrupt lower priority tasks.
  - The strategy provides the lowest possible latencies for the highest priority events.
    - Lower priority events still suffer from starvation and are at risk for missing their deadlines.
- Adds complexity requiring an interrupt controller to trigger events.

Preemptive Strategy Summary

- Event prioritization is important.
  - If all events have the same priority, the design behaves like priority polling.
  - Interrupts without priority have added complexity and no performance advantage.
- Prioritization strategy: shortest period = highest priority.
  - When interrupts fully enabled, highest priority event latency is very small.
  - Calculating response time of lower priority interrupts becomes difficult.
- Due to shared data and resources, we may need to disable interrupts.
  - When interrupts are disabled for the entire ISR, the design performs the same as priority polling.
  - The latency of all ISRs is affected when interrupts are disabled.
    - Latency of all events increases by the maximum time that interrupts are disabled.
    - It is very important to keep the time interrupts are disabled as short as possible.

Even small changes make a big difference.



```
    //loop for testing the effect of disabling interrupts
    while (true) {
        IntMasterDisable();
//        delay_us(100);
        count_unloaded++;
        count_loaded--;
        IntMasterEnable();

        IntMasterDisable();
//        delay_us(200);
        IntMasterEnable();
    }
#endif
```

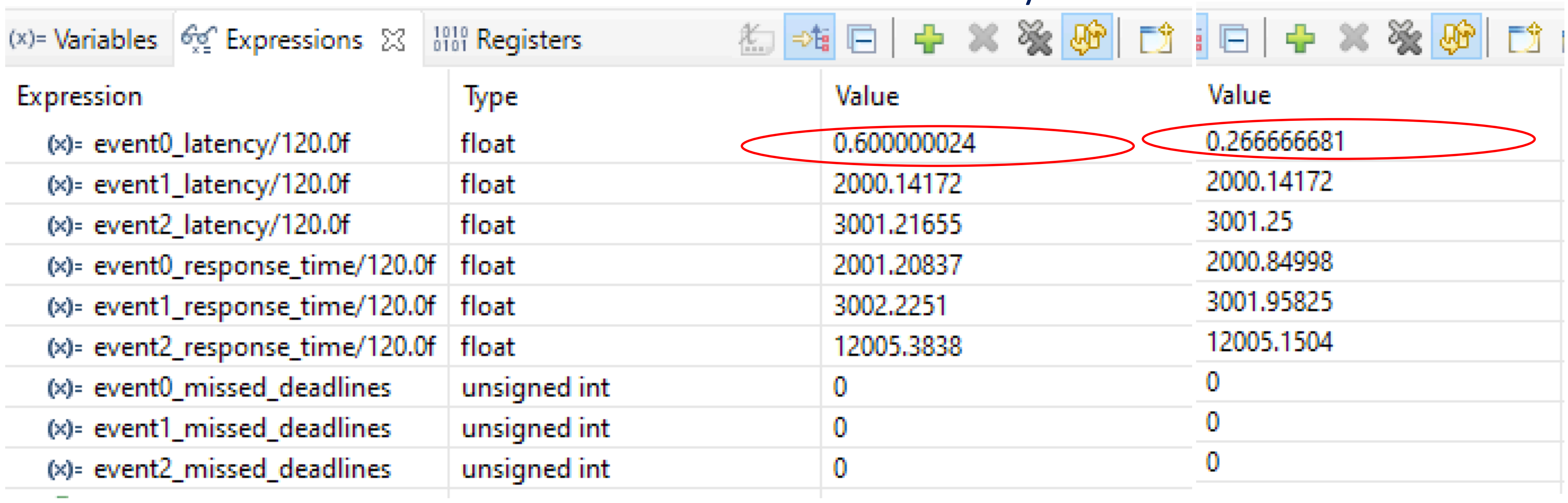
Original foreground while loop was empty.

Now we disable interrupts, for just two operations.

- It more than doubled the latency of the highest priority event, 600 nsec >> 266 nsec.

New latency

Original Latency without disabling ISRs



Expression	Type	Value	Value
(x)= event0_latency/120.0f	float	0.600000024	0.266666681
(x)= event1_latency/120.0f	float	2000.14172	2000.14172
(x)= event2_latency/120.0f	float	3001.21655	3001.25
(x)= event0_response_time/120.0f	float	2001.20837	2000.84998
(x)= event1_response_time/120.0f	float	3002.2251	3001.95825
(x)= event2_response_time/120.0f	float	12005.3838	12005.1504
(x)= event0_missed_deadlines	unsigned int	0	0
(x)= event1_missed_deadlines	unsigned int	0	0
(x)= event2_missed_deadlines	unsigned int	0	0

Better Analysis Methods

- When we introduced interrupts, response time of lower priority events becomes difficult to calculate.
    - Often they are preempted multiple times?
    - What if conditions change and event 1 can be preempted multiple times too.

Event	Period	Execution Time	Latency	Response Time (Latency + ?)	Relative Deadline (Period)	Schedulable ? YES = response < deadline
event0	6 ms	2 ms	0 ms	2 ms	6 ms	Yes
event1	8 ms	1 ms	2 ms	3 ms Tricky to calc	8 ms	Yes
event2	12 ms	6 ms	3 ms	12 ms Tricky to calc	12 ms	Maybe, marginal

- We need a more robust method of calculating response time to guarantee we have thought of the worst case scenario.

Rate-Monotonic Scheduling Theory

- Rate-Monotonic Scheduling (RMS) Theory provides a more reliable way to verify the scheduling.
- The system must satisfy the following conditions.
  - All real-time tasks (events) are periodic with fixed periods.
    - Relative deadline = event period.
  - Fixed priorities are assigned to events according to period.
    - Shortest period = higher priority.
  - Higher priority tasks preempt lower priority ones.
  - Execution time per event is fixed for each task.
    - This is accomplished by using the maximum value.
  - Task switching overhead is negligible.
  - Tasks do not synchronize with each other
    - No blocking or disabling interrupts.

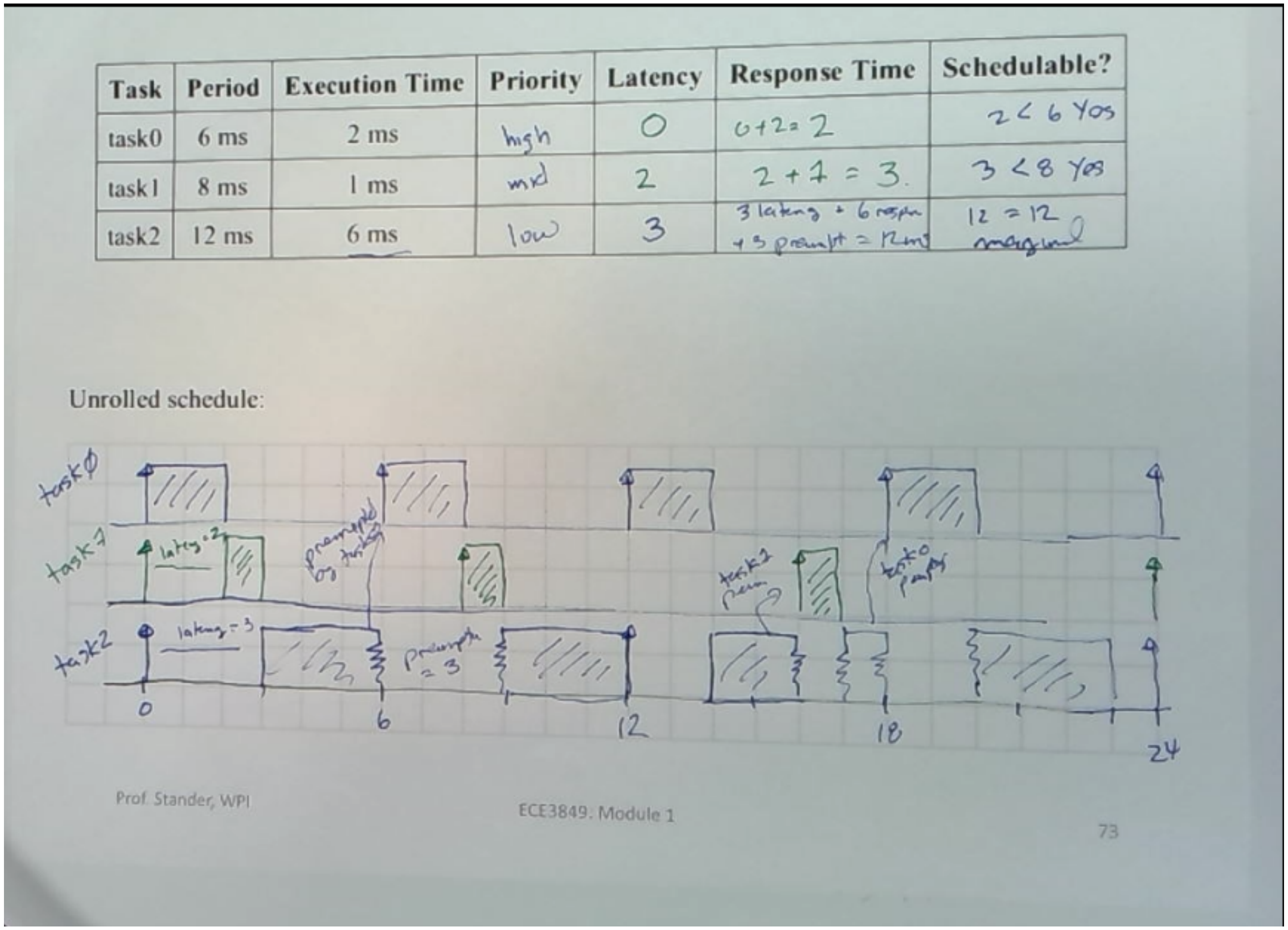
RMS – Theorem

- Theorem: “The first deadline is the hardest”.
- If a task meets its first deadline when all tasks are released at the same time, then it will meet all future ones.
  - Worst case when all the tasks happen at once.
- Look at each task.
  - Does it meet its first deadline?
  - If yes, it is schedulable for all conditions.
- Lets use a graphical model.

Task	Period	Execution Time	Priority	Latency	Response Time	Schedulable?
task0	6 ms	2 ms				
task1	8 ms	1 ms				
task2	12 ms	6 ms				

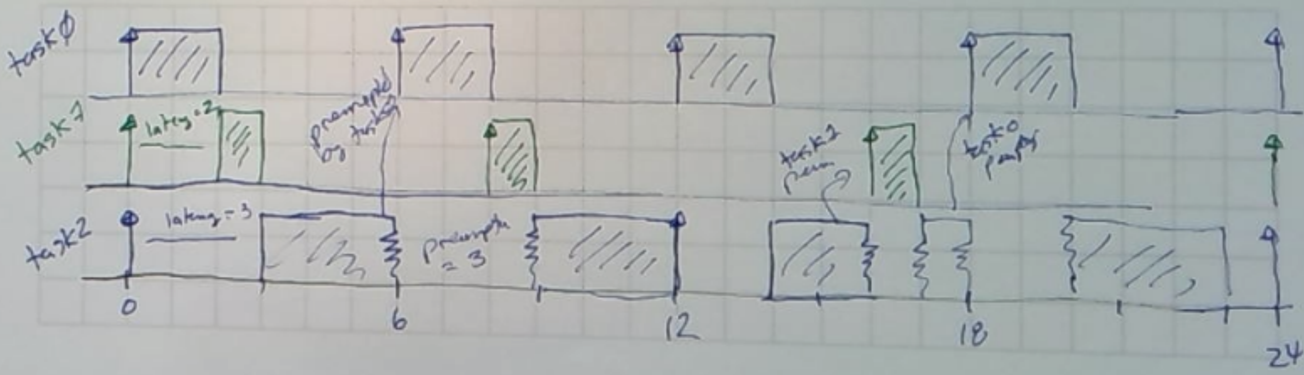
Unrolled schedule:

In-Class Worksheet



Task	Period	Execution Time	Priority	Latency	Response Time	Schedulable?
task0	6 ms	2 ms	high	0	0+2= 2	2 < 6 Yes
task1	8 ms	1 ms	mid	2	2 + 1 = 3.	3 < 8 Yes
task2	12 ms	6 ms	low	3	3 latency + 6 respn + 3 preempt = 12ms	12 = 12 marginal

Unrolled schedule:



Prof. Stander, WPI

ECE3849: Module 1

73

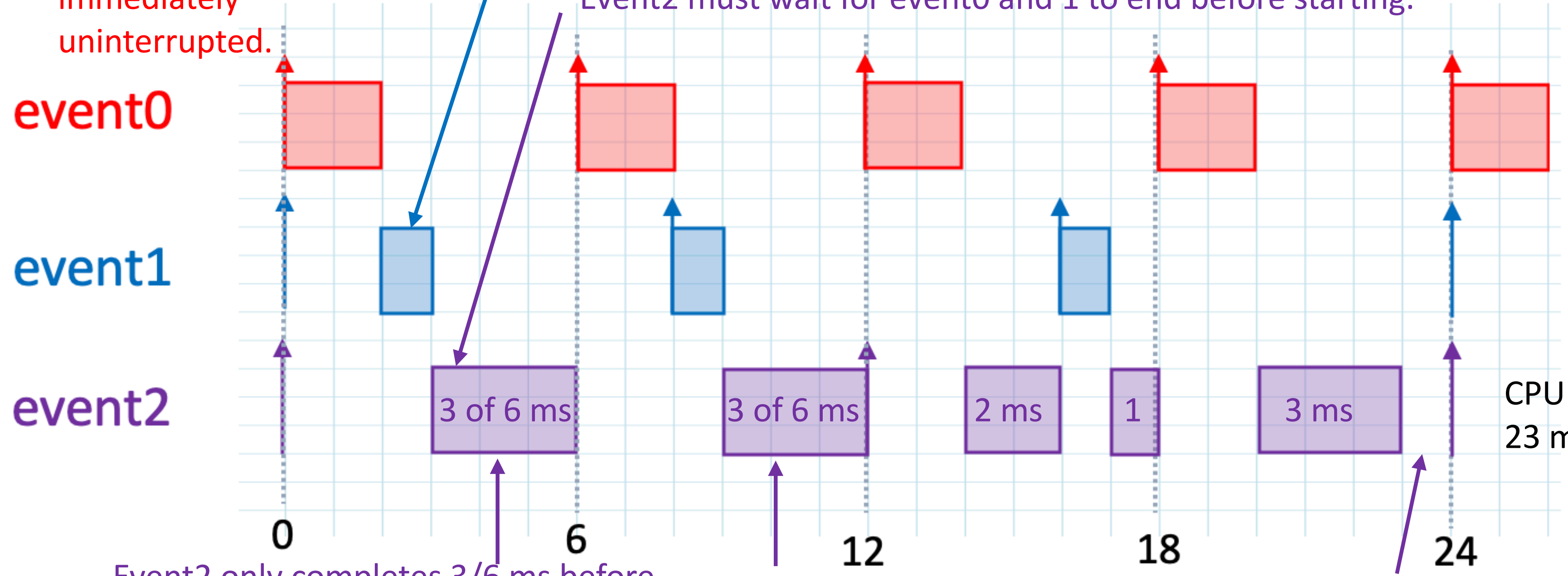
RMS Graphical Model

Event	Period	Execution Time	Priority	Latency	Response Time (Latency + ?)	Relative Deadline (Period)	Schedulable ? YES = response < deadline
event0	6 ms	2 ms	High	0 ms	2 ms	6 ms	Yes
event1	8 ms	1 ms	Mid	2 ms	3 ms	8 ms	Yes
event2	12 ms	6 ms	Low	3 ms	12 ms	12 ms	Yes, marginal

Event0 happens like clock work, starts immediately uninterrupted.

Event1 must wait for event0 to end before starting.

Event2 must wait for event0 and 1 to end before starting.



CPU is running tasks 23 msec of 24 msec.

Event2 only completes 3/6 ms before event0 and then 1 preempts it.

Event2 resumes when event 0 and 1 are complete.

First deadline 0 ms margin. Second deadline easier has 1 msec.

Response-Time Analysis

- The graphical method is difficult for non-round number and as the number of tasks grow.
- Given the execution time and period of each task, the response time can be calculated using iterative equation.

T_i = period of task i

E_i = execution time per event of task i

R_i = response time of task i

i = 1 for high priority task

i = 2 for mid priority task

i = 3 for low priority task

- Initialize the response time.

$$R_i^1 = \sum_{j=1}^{i} E_j$$

$R^1_1 = E_1$

$R^1_2 = E_1 + E_2$ (best case assumes task 1 only interrupts task 2 once)

$R^1_3 = E_1 + E_2 + E_3$ (best case assumes task 1 and task 2 only interrupt task 3 once)

- Repeat this equation until it converges, where k is the number of iterations.

$$R_i^{k+1} = E_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^k}{T_j} \right\rceil E_j$$

$R^2_1 = E_1$

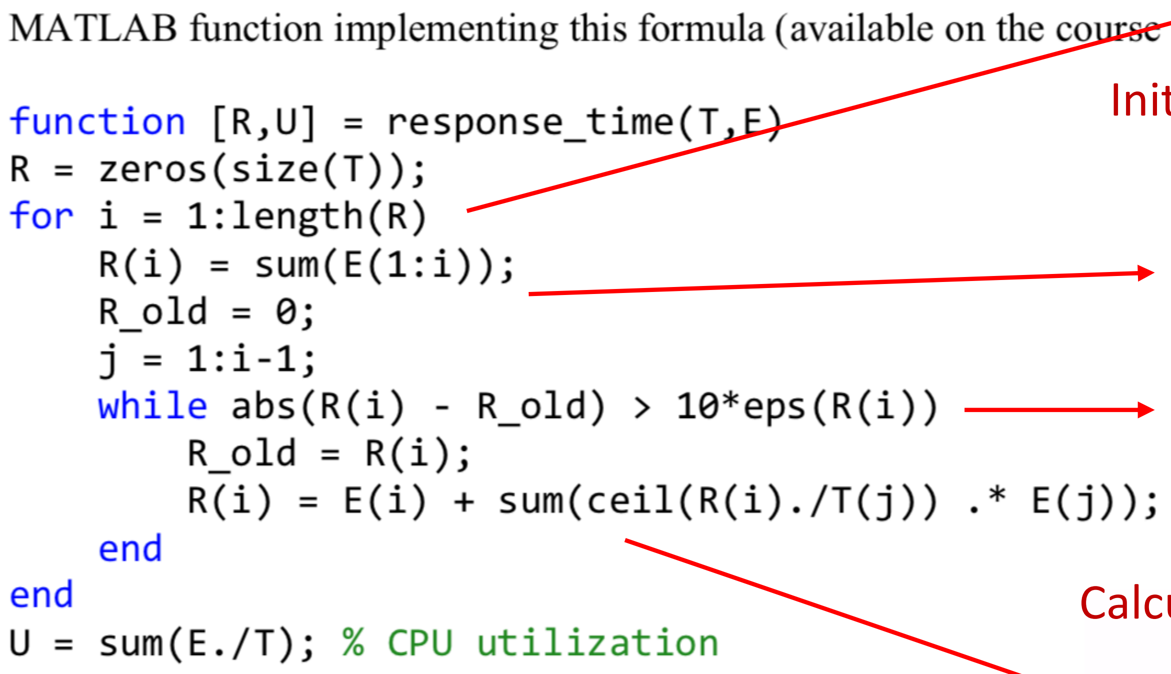
$R^2_2 = E_2 + \text{ceil}(R^1_2/T_1)* E_1$

$R^2_3 = E_3 + \text{ceil}(R^1_3/T_1)\ E_1 + \text{ceil}(R^1_3/T_2)\ *E_2$

Adjustment for multiple interrupts

Response-Time Analysis: Matlab

- This function is implemented in Matlab.



MATLAB function implementing this formula (available on the course website):

```
function [R,U] = response_time(T,E)
R = zeros(size(T));
for i = 1:length(R)
    R(i) = sum(E(1:i));
    R_old = 0;
    j = 1:i-1;
    while abs(R(i) - R_old) > 10*eps(R(i))
        R_old = R(i);
        R(i) = E(i) + sum(ceil(R(i)./T(j)) .* E(j));
    end
end
U = sum(E./T); % CPU utilization
```

Loops through each task.

Initializes the response time.

$$R_i^1 = \sum_{j=1}^{i} E_j$$

Loops until the difference between loops is very small.

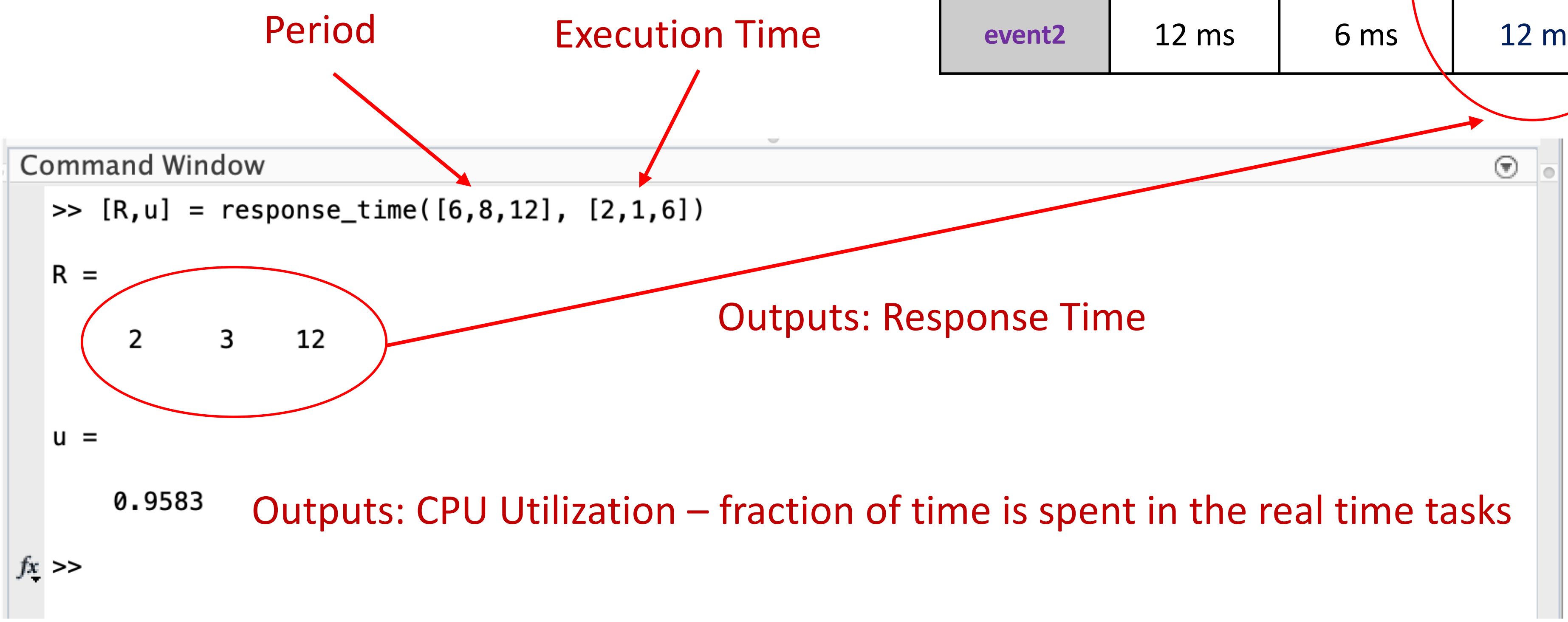
Calculates response time.

$$R_i^{k+1} = E_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^k}{T_j} \right\rceil E_j$$

Matlab Example Run

- Matlab file is named "response_time.m".
- The file is in the search path or current directory in Matlab.

Event	Period	Execution Time	Response Time
event0	6 ms	2 ms	2 ms
event1	8 ms	1 ms	3 ms
event2	12 ms	6 ms	12 ms


Period
Execution Time
Command Window
Outputs: Response Time
Outputs: CPU Utilization – fraction of time is spent in the real time tasks

```
>> [R,u] = response_time([6,8,12], [2,1,6])

R =

     2     3    12


u =

    0.9583

>>
```

Theorem: Processor Utilization

- CPU utilization defines what fraction of the time is spent performing tasks.
  - It can be calculated by summing the fraction of time each task executes in its given period.

$$U = \sum_{i=1}^{n} \frac{E_i}{T_i}$$

where n is the number of task
E is the execution time.
T is period of task.

- CPU Utilization in previous example.
  - U = 2/6 + 1/8 +6/12
  - U = 23/24 = 0.958

Event	Period (T)	Execution Time (E)	
event0	6 ms	2 ms	2 ms
event1	8 ms	1 ms	3 ms
event2	12 ms	6 ms	12 ms

Processor Utilization Upper Bound

- Given the RMS assumptions an upper bound for utilization is derived given a certain number of tasks.
  - Where n is the number of tasks.
  - If utilization is less than this, it can definitely be scheduled.
  - If utilization is more, that it may still be possible but further analysis is needed.
    - The last example had 3 tasks with a 0.958 utilization which is greater than the 0.7798 upper bound but it was schedulable.
  - In practice, we can not use 100% of the CPU for real time events. Generally it is good to have 10 – 20% margin.

$$U \leq n\left(2^{1/n} - 1\right)$$

n	*max(U)*	*max(U), %*
1	1	100%
2	0.8284	82.84%
3	0.7798	77.98%
4	0.7568	75.68%
5	0.7435	74.35%
∞	0.6931 = ln(2)	69.31%