Embedded Systems II, Assignment 5
Mälardalen University

Emil Broberg
School of Innovation, Design and Engineering
Mälardalen University, Västerås, Sweden
Email: ebg20007@student.mdu.se

Question 1

a)

Cache coherency refers to the property of consistently storing data in multiple caches that replicate the same data. In a multi-core or multiprocessor system with a shared cache, each core or processor has a copy of the same data. When one core or processor modifies the data, the other cores or processors need to be notified of the change and update their copies of the data. If all processors have a consistent view of the data at each memory location and at each point in time, then the system is said to be cache coherent.

b)

A shared memory can be beneficial in a multi-core stystem. It allows different cores to access a common pool of memory, facilitating communication by reading and writing data to shared locations. However, it can also be problematic if the system gets too large. If the system gets to large issues can arise such as larger overhead because of cores waiting for access to the shared memory and communication via shared memory. If no proper synchronization is used, the system might become incoherent and also lead to race conditions.

c)

The two main scheduling approaches are partitioned and global scheduling.

Partitioned: In partitioned scheduling the tasks and preocessors are patitioned into groups. This means that each preocessor get a set of tasks that wait in a queue dedicated to that preocessor. This makes each preocessor operate independently which increase simplicity of the system.

- **Strengths:**
  - **Simplicity:** Each preocessor operates independently and scheduling can be done locally.
  - **Predictability:** Since each preocessor operates independently and scheduling is done locally, the system can be designed in a predictable way.
  - **Low overhead:** Since each preocessor independently manages its own tasks, there is minimal overhead associated with coordination and communication between preocessors.
- **Weak points:**
  - **Load imbalance:** If the tasks are not evenly distributed between the preocessors, some preocessors might be idle while others are overloaded.

Global: In global scheduling, all preocessors share a common queue of tasks. A global scheduler makes decisions about task allocation, considering the entire set of tasks and the status of each preocessor. Tasks can be distributed between processors as needed.

- **Strengths:**
  - **Adaptive:** The tasks are assigned dynamically and the system can adapt to changes in the workload. In an open system, a task can be added or removed from the system dynamically and the global scheduler can adapt to this.
  - **Less context switching:** Since the tasks are assigned dynamically, there is less context switching between tasks. This reduces the overhead of the system. Tasks can only get preempted once all processors are working on a task.
- **Weak points:**
  - **Complexity:** The global scheduler needs to consider the entire set of tasks and the status of each preocessor when making scheduling decisions. This increases the complexity of the system.
  - **RM and EDF not optimal:** RM and EDF might not be optimal in a global scheduling system. Implementing global scheduling can be more complex compared to partitioned scheduling. Coordinating tasks across processors will require a more complex scheduling design.

– **Overhead:** The global scheduler needs to communicate with all preocessors to make scheduling decisions. This increases the overhead of the system.

Question 2

Constraints

- **Low cost:** The cost of the system is an important constraint. The cost of the system should be as low as possible if large quantities will be produced.
- **Power consumption:** The power consumption of the system is an important constraint which has to be considered. If an embedded system is battery powered, the power consumption of the system should be as low as possible to make sure that the functionality of the system will be available for as long as possible. If the system is not battery powered, the power consumption of the system should be as low as possible to reduce the heat generated by the system.
- **Performance:** There might be performance constraints on the system. This might be to enhance user experience, make sure safety critical functions operate as soon as needed or to make sure that the system can handle the workload.
- **Size:** Size can be an important constraint since many embedded systems have to fit in tight spaces, such as electronic systems in cars, phones, planes, etc. Some embedded systems might also have to be portable, such as a hand held communication radio, headphones and calculators.

Functionality

The intended functionality of the embedded system has to be clearly defined. This is importand to know to decide on what scheduling algorithms to use, what hardware to use, what software to use, etc. The functionality of the system might also be constrained by the hardware and software used. For example, if the system is battery powered, the functionality of the system might be constrained by the power consumption of the system. If the system is a safety critical system, the functionality of the system might be constrained by the performance of the system.

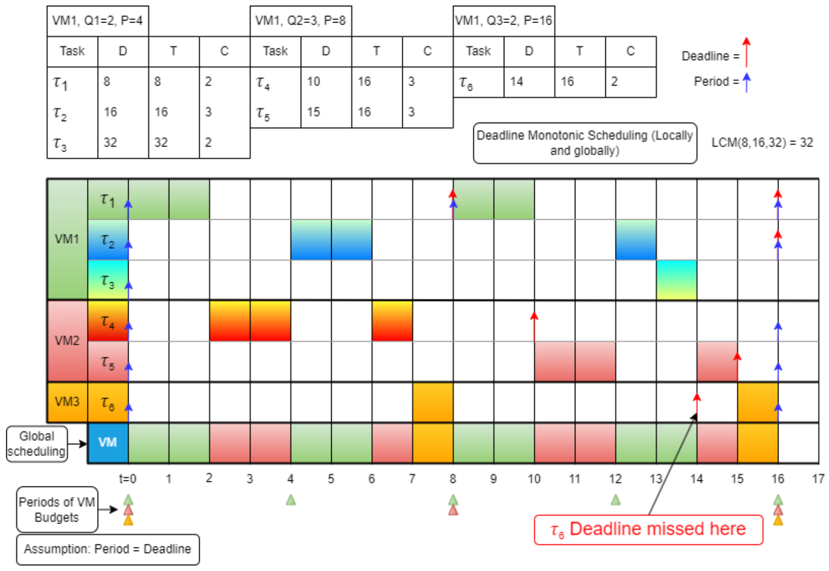
Real-time requirements

If there are real-time requirements, the system have to be designed to execute certain tasks in a deterministic manner and continually adapt to changes in the environment surrounding the system.

Question 3

a)

The following figure will show the schedule for the virtual machines and their local task sets when using the *Deadline Monotonic* scheduling algorithm locally and globally. When scheduling globally I assume that the deadline, which is used to determine priority, is equal to the period of the VM interface.



VM1, Q1=2, P=4

Task	D	T	C
τ_1	8	8	2
τ_2	16	16	3
τ_3	32	32	2

VM1, Q2=3, P=8

Task	D	T	C
τ_4	10	16	3
τ_5	15	16	3

VM1, Q3=2, P=16

Task	D	T	C
τ_6	14	16	2

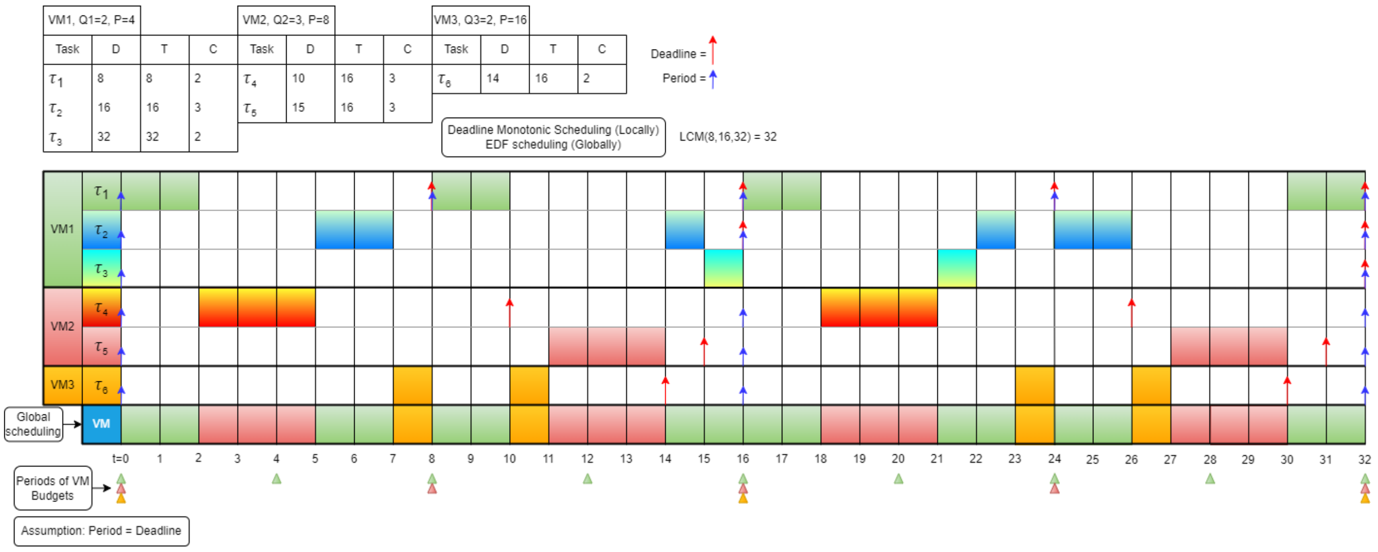
Deadline =
Period =
Deadline Monotonic Scheduling (Locally and globally)
LCM(8,16,32) = 32
VM1
VM2
VM3
τ_1
τ_2
τ_3
τ_4
τ_5
τ_6
Global scheduling
VM
t=0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Periods of VM Budgets
Assumption: Period = Deadline
τ_6 Deadline missed here

Figure 1. Schedule for the virtual machines and their local task sets when using the *Deadline Monotonic* scheduling algorithm locally and globally.

As we can see in figure 1 the deadline of τ_6 is missed at $t = 14$.

b)

The following figure will show the schedule for the virtual machines and their local task sets when using the *Earliest Deadline First* scheduling algorithm globally, but still *Deadline Monotonic* locally. When scheduling globally I still assume that the deadline, which is used to determine priority, is equal to the period of the VM interface.



VM1, Q1=2, P=4			
Task	D	T	C
τ_1	8	8	2
τ_2	16	16	3
τ_3	32	32	2

VM2, Q2=3, P=8			
Task	D	T	C
τ_4	10	16	3
τ_5	15	16	3

VM3, Q3=2, P=16			
Task	D	T	C
τ_6	14	16	2

Deadline =
Period =
Deadline Monotonic Scheduling (Locally)
EDF scheduling (Globally)
LCM(8,16,32) = 32
VM1
VM2
VM3
τ_1
τ_2
τ_3
τ_4
τ_5
τ_6
Global scheduling
VM
t=0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Periods of VM Budgets
Assumption: Period = Deadline

Figure 2. Schedule for the virtual machines and their local task sets when using the *Earliest Deadline First* scheduling algorithm globally, but still *Deadline Monotonic* locally.

As we can see in figure 2 all deadlines are met both globally and locally.

c)

There are a few things that can be done, for example we can follow the example above of trying different scheduling algorithms, locally and globally, and analyse the traces to find the best option for that specific case. What more can be done is to use offsets when scheduling tasks. This can mitigate missing deadlines when loads are high, especially for lower priority tasks. So, in our example in question *a* it could be a good idea to use offsets to give room for τ_6 to execute.