KTH ROYAL INSTITUTE OF TECHNOLOGY

Multiprocessor Scheduling

Ingo Sander and Matthias Becker
Contact: ingo@kth.se

Liu: Chapter 9

IL2212 – Embedded Software

1

Outline

- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

IL2212 – EMBEDDED SOFTWARE 2

2

Outline

- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

3

Multiprocessor Systems

- We switch our focus to multiprocessor systems
  – Principles we learned give a solid base to tackle these systems
  – New problems arise
    - Task assignment problem
    - Multiprocessor protocols for resource access control
    - Interprocessor synchronization (not discussed here)
- Again: overall goal is to meet all deadlines

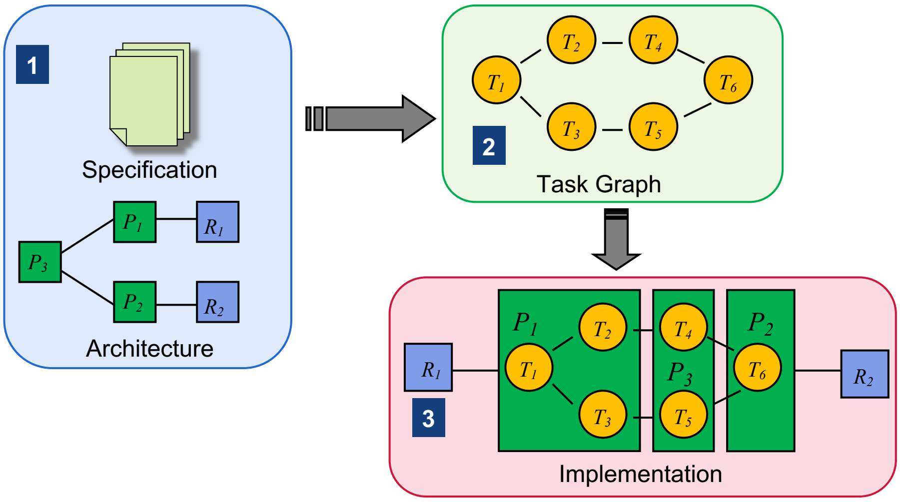
5

Multiprocessor Systems

- Embedded systems often consist of several processors
  - Processors can be homogeneous (same type) or heterogeneous (different type)
    - Also, hardware units performing a dedicated processing function (like an FIR-filter) can be viewed as processors
  - Embedded systems are often distributed
    - In cars several processors are connected by a bus

6

Simplified Design Flow



7

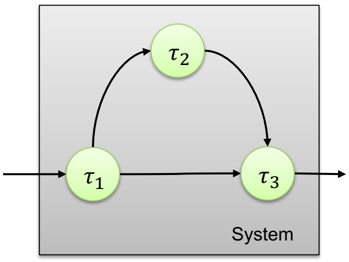
What can be parallelized?

- Multiprocessors allow for parallel execution
- But not all applications can be parallelized

8

Concurrent Activities

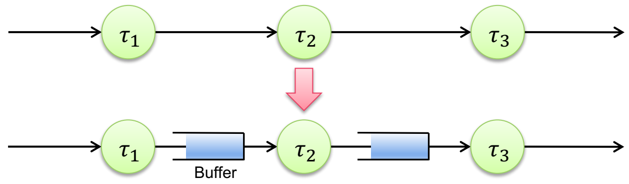
- Concurrent activities can be assigned to different processors
- τ_1, τ_2, and τ_3 can run on different processors



9

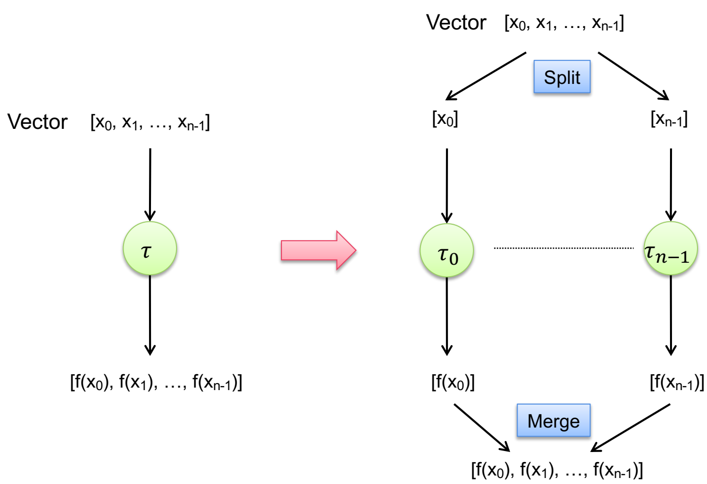
Pipeline

- Sequential programs that comply to data flow style can be parallelized using pipelining
  – Streaming Media Applications
- τ_1, τ_2, and τ_3 can run on different processors, in case the result of each process only depends on the input data
  – no global variables
  – buffers needed to enable independent execution



10

Data-Parallelism



11

Simplified Design Flow

- The design of a multiprocessor embedded system is very challenging
  - architecture may not be fixed
  - many different implementations possible
  - remote access to resources
  - synchronization required
  - precedence constraints
- The design of such a system must be based on well-understood principles and techniques

12

Outline

- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

13

Multiprocessor Scheduling

- A single core scheduler has to answer the question:

  Which task to execute?

- A multiprocessor scheduler has to answer the question:

  Which task to execute? And where?

  Task assignment becomes important!

Allocation Problem

Allocation of tasks to processors falls into 3 categories:

1. No migration is allowed: All jobs of a task execute on the same processor
2. Task-level migration: Different jobs can execute on different processors, but each job can only execute on one processor
3. Job-level migration: Each job can migrate between processors but can never execute on multiple processors at the same time

Priority Assignment

Assignment of priorities:

1. Fixed task priority: A fixed priority is assigned to all jobs of a task
2. Fixed job priority: Different jobs of a task may have different priorities but the same job has a single static priority
3. Dynamic priority: The priority of the job may change over its lifetime

16

Fundamental limitation

”…the utilization guarantee bound for any static-priority multiprocessor scheduling algorithm (partitioned or global) cannot be higher than 1/2 of the capacity of the multiprocessor platform. “
– Anderson, Baruah, Jonsson, Real-Time Systems Symposium 2001.

17

Outline

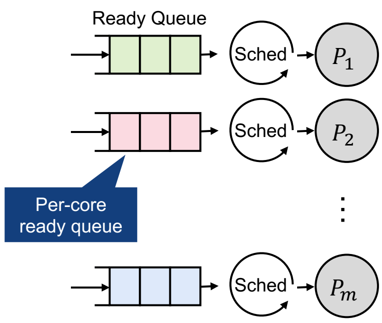
- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

18

Partitioned Scheduling

When no migration is allowed by the scheduling algorithm, the algorithm is referred to as partitioned.

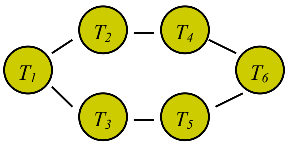
- Each core has its own set of tasks (assigned offline)
- Cores can schedule their tasks independent of each other



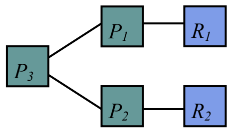
19

Task Assignment

- In a partitioned (i.e. static) system, the application is partitioned into modules that are bound to processors
- This is called task assignment



Task Graph



Architecture

Task Assignment

- At some stage in the design process, the execution times resource requirements, data and control dependencies of all the tasks become known
- Task assignment determines
  - how many processors are needed
  - on which processor each task executes

Task Assignment

- Task assignment is a very complex (NP-hard) problem
  - often heuristics are used
  - task assignment is most often done off-line
  - in a dynamic system, task assignment can be used as an acceptance test

Task Assignment

- The simplicity of the model in the task assignment problem can vary in complexity
  1. Communication and placement of resources is ignored
  2. Only communication costs are considered
  3. Both communication costs and resource access costs are considered
- All models have their merits in the design flow

Task Assignment Based on Execution-Time Requirements

- It is often meaningful to ignore resources and communication in an early design phase
  - fits to often used assumption that tasks in a real-time system are independent
- In some shared-memory applications with few memory conflicts, the communication costs may be very small

Task Assignment Based on Execution-Time Requirements

- Task Assignment Problem
  - Given are the utilizations of n tasks
  - The system shall be partitioned into modules (set of tasks) in such a way that the tasks in each module are schedulable by themselves on a processor according to a uniprocessor scheduling algorithm of a given class
  - The task assignment is defined by a subset of tasks in every module

Task Assignment Based on Execution-Time Requirements

- Task assignment problem can be formulated as the simple uniform-size bin-packing problem
- Sizes of all bins is schedulable utilization U_{LUB} of the algorithm (EDF = 1; RMA = $ln\ 2$)
- Sizes of each item (task) is the utilization
- The number of bins required to pack the items is the number of processors required to feasibly schedule all tasks

26

First Fit Algorithm

First-Fit is a simple heuristic algorithm for the bin-packing problem

- Tasks are assigned one by one in arbitrary order
- First task is assigned to processor P_1
- After i-1 tasks, the i-th task τ_i is assigned to processor P_k,
  - if the total utilization of τ_i and the tasks already assigned to P_k is equal to or less than the schedulable utilization U_{LUB}
  - and assigning τ_i to any of the processors $P_1, P_2, \ldots, P_{k-1}$, would make the total utilization of tasks on the processor larger than U_{LUB}

27

Variable-Size Bin-Packing

For the EDF algorithm U_{EDF} does not depend on the number of tasks

But in the RM algorithm U_{RM} depends on the number of tasks

- If a fixed size of a bin is assumed, we have to use $U_{RM} = \ln 2$
- Otherwise, bin-size can be a function of the number of tasks
  - increasing the complexity of the problem!

Task Assignment To Minimize Total Communication Costs

Communication costs between tasks which run on same processor is usually significantly lower than for tasks that run on different processors

- Communication via registers possible on single processor
- Communication via shared bus and shared memory on shared memory multiprocessor
- Communication via network on network-on chip

Task Assignment To Minimize Total Communication Costs

Cost of communication:

- Abstract parameter that captures indirectly communication time
- Cost of communication depends on the volume of data exchanged and the bandwidth of the communication link
- Here we only discuss systems of homogeneous processors, which communicate via a shared communication channel (e.g. a bus)

Task Assignment To Minimize Total Communication Costs

When communication costs are considered objective of task assignment is twofold

- Find minimum number of processors needed
- Find minimum of communication costs for this number of processors

Task Assignment To Minimize Total Communication Costs

The *communication cost* between two tasks τ_i and τ_k is expressed by a single value $C_{i,k}$, if the tasks are placed on different modules

$C_{i,k}$ usually reflects the amount of data exchanged between both tasks

To account for memory contention an *interference cost* can be included if tasks τ_i and τ_k are placed on the same processor

Task Assignment To Minimize Total Communication Costs

In order to find a feasible solution different optimization techniques like heuristic algorithms or constraint programming can be used

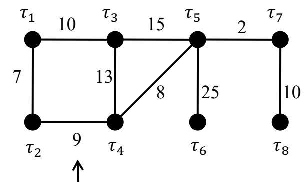
Given a cost function, minimize

1. the number of processors needed
2. the total communication cost

Example of Task Partitioning

i	τ_i	U_i	i	τ_i	U_i
1	(2,1)	0.50	5	(6,1)	0.17
2	(3,1)	0.33	6	(10,1)	0.10
3	(4,1)	0.25	7	(15,1)	0.07
4	(5,1)	0.20	8	(25,1)	0.04

(EDF scheduling)



Communication Cost

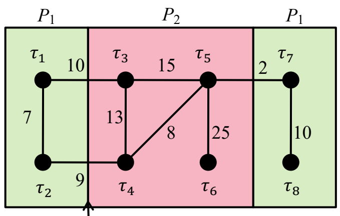
Objective is to find a partitioning, which is feasible at minimal costs (here interference cost is neglected!)

40

Example of Task Partitioning

i	τ_i	U_i	i	τ_i	U_i
1	(2,1)	0.50	5	(6,1)	0.17
2	(3,1)	0.33	6	(10,1)	0.10
3	(4,1)	0.25	7	(15,1)	0.07
4	(5,1)	0.20	8	(25,1)	0.04

(EDF scheduling)



Communication Cost

1. Utilization: $U(P_1) = 0.94$, $U(P_2) = 0.72$
2. Communication Cost: C = 10 + 9 + 2 = 21

This is just one possible solution. Not necessarily the best one!

41

Complexity of problem can easily increase…

- The problem can easily become even more complex, if
  - heterogeneous processing units are allowed
  - communication cost depends on the mapping (network-on-chip, cost depends on the distance between two nodes)
  - Processing units have additional parameters like frequency/voltage scaling and other objectives like power efficiency are also targeted

42

Outline

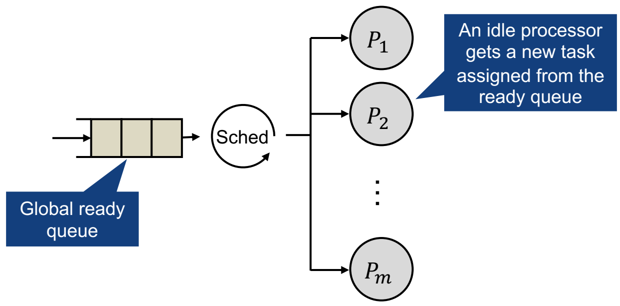
- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

43

Global Scheduling

When job-level migration is allowed by the scheduling algorithm, the algorithm is referred to as partitioned.

- There is only one ready queue for all processors
- Assignment to a processor is done online



Dahll's Effect

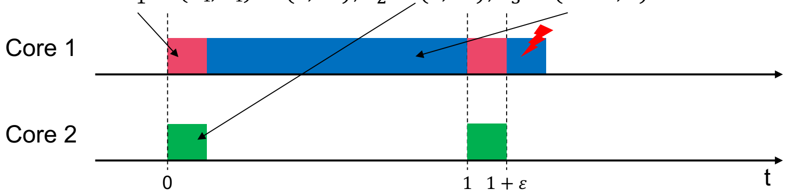
Identified by Dhall and Liu in their seminal paper (Oper. Res. 1978)

Assumption:

- Global Scheduling
- Periodic Tasksets with implicit deadlines

Example:

- $\tau_1 = (T_1, C_1) = (1, 2\varepsilon)$, $\tau_2 = (1, 2\varepsilon)$, $\tau_3 = (1 + \varepsilon, 1)$



Dahll's Effect

$$U_{global} = m\frac{2\varepsilon}{1} + \frac{1}{1+\varepsilon}$$

when $\varepsilon \to 0$

- This affects RM, DM, and EDF
- Cause:
  - Access to the processors is decided based on the periods/deadlines but not based on the computation demands!

Global vs. Partitioned Scheduling

	Partitioned Scheduling	Global Scheduling
+	• High degree of predictability • Easy to implement • Small overheads at runtime • Single processor scheduling can be reused	• Offline mapping problem is avoided • Fewer context switches
-	• Task allocation problem • NP-Hard	• Migration costs are expensive • Previously optimal algorithms are not optimal anymore

Outline

- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

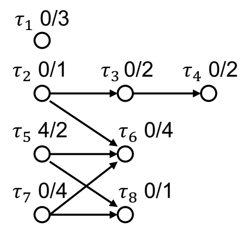
Multiprocessor Scheduling Anomalies

Counter intuitive effects when changing parameters of a taskset are referred to ask scheduling anomaly.

→ Improving the system may lead to an unschedulable system, even it it was schedulable before!

Example: Priority-Driven Scheduling

- Given are eight tasks with the following priorities: $\pi_1 > \pi_2 > \ldots > \pi_8$
- Jobs are scheduled on two processors P_1 and P_2
- Communication cost is negligible
- There is only one common priority queue

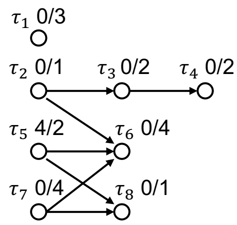


a/C = Arrival Time / Execution Time

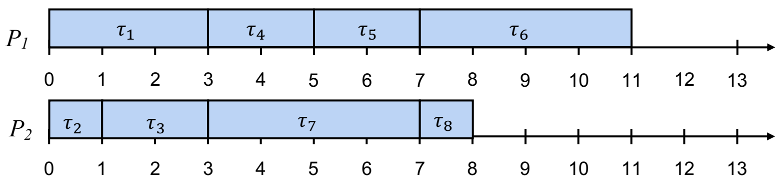
52

Example: Priority-Driven Scheduling

a/C = Arrival Time/Execution Time

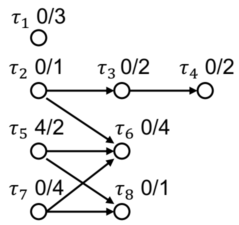


Non-preemptive schedule



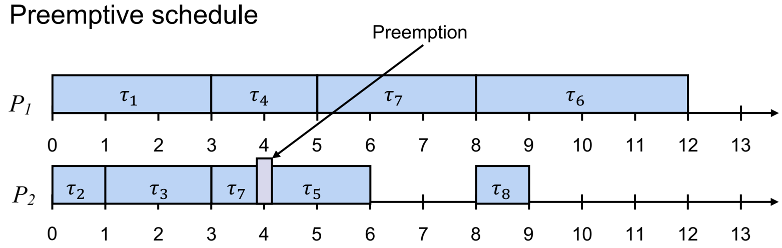
53

Example: Priority-Driven Scheduling



a/C = Arrival Time/Execution Time

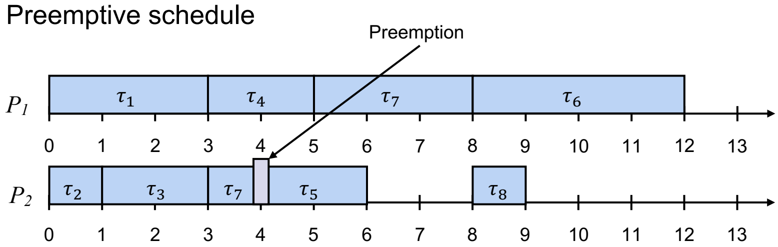
Preemptive schedule



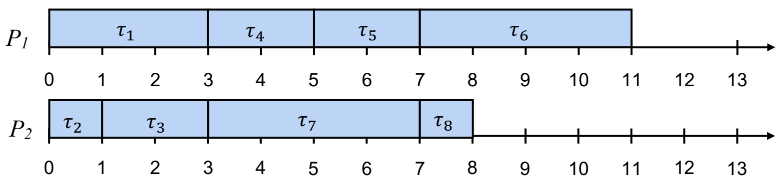
54

Example: Priority-Driven Scheduling

Preemptive schedule



Non-preemptive schedule



55

Example: Priority-Driven Scheduling

Non-preemptive schedule gave better performance than preemptive schedule

- No general rule, but priority scheduling results can be non-intuitive
- More examples in Chapter 2.4 in [Buttazzo, 2011]

If a task set is optimally scheduled on a multiprocessor with some priority assignment, a fixed number of processors, fixed execution times and precedence constraints, then increasing the number of processors, reducing execution times, or weakening the precedence constraints can increase the schedule length. [Graham, 1976]

56

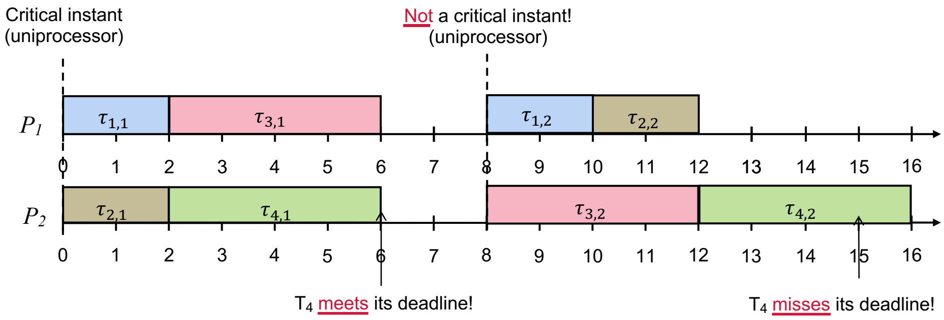
Critical Instant Effect

For fixed-priority scheduling on multi-processors the critical instant does not always result if all higher priority tasks arrive at the same time. [Lauzac et al.1998]

57

Critical Instant Effect

- Format: τ_i=(ϕ_i, T_i, C_i, D_i) – Deadline Monotonic Scheduling
  - τ_1 =(0,8,2,2), τ_2 =(0,10,2,2), τ_3 =(0,8,4,6), τ_4 =(0,8,4,7)



58

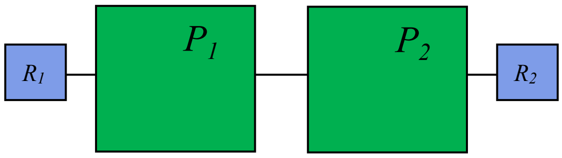
Outline

- Background
- Multiprocessor Scheduling
  - Fundamental Results
  - Partitioned Scheduling
  - Global Scheduling
  - Scheduling Anomalies
- Resource Sharing

61

Local vs. Remote Resources

- We assume that each resource resides on a processor
  - Scheduler of that processor controls the access to the resource
  - If a task is using the resource, the critical section is executing on the processor belonging to the resource



62

Local vs. Remote Resources

Multiprocessor Priority-Ceiling Protocol (MPCP) Resource Model

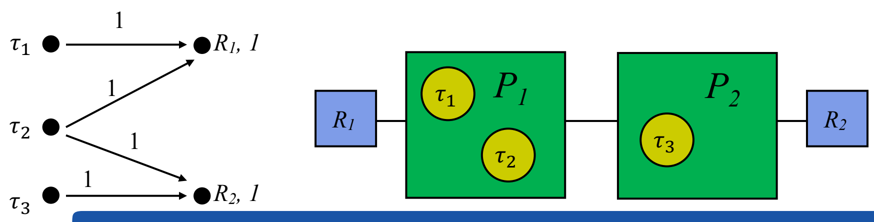
- The processor on which each resource resides is called its *synchronization processor*
- The processor on which each task is released and becomes ready is called the *local processor* of the job
- A resource that resides on the local processor is a *local resource*
- A resource that resides on another processor is a *global resource*
- A *global resource* is a resource that is required by jobs that have different local processors

63

Local vs. Remote Resources

MPCP

- P_1 is the synchronization processor of R_1
- P_2 is the synchronization processor of R_2
- τ_1 and τ_2 are local jobs on P_1 and τ_3 is a local job on P_2

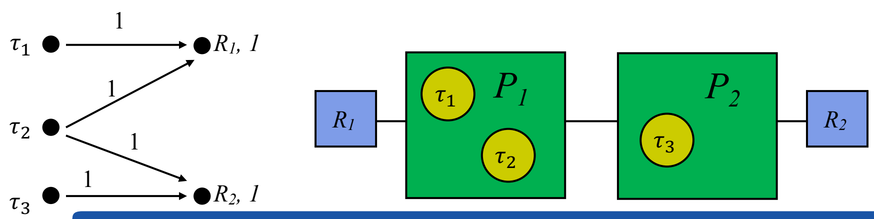


64

Local vs. Remote Resources

MPCP

- Since τ_1 uses the remote resource R_2 on P_2, R_2 is a global resource
- During the remote global critical section while τ_2 uses the resource R_2, τ_2 executes on P_2
- When the global critical section of τ_2 completes, the job τ_2 returns to P_1

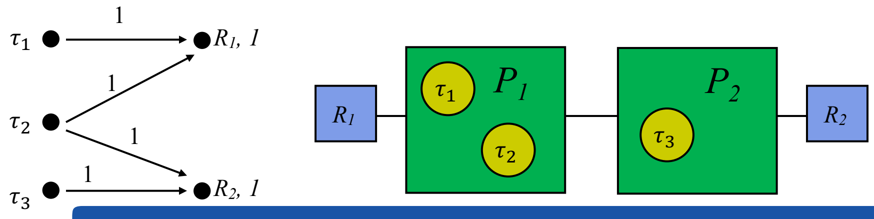


65

Local vs. Remote Resources

End-to-End Resource Model

- The task τ_2 is an end-to-end task and consists of three component tasks
  - First component executes on P_1, the remote critical section executes on P_2 and the third component executes on P_1

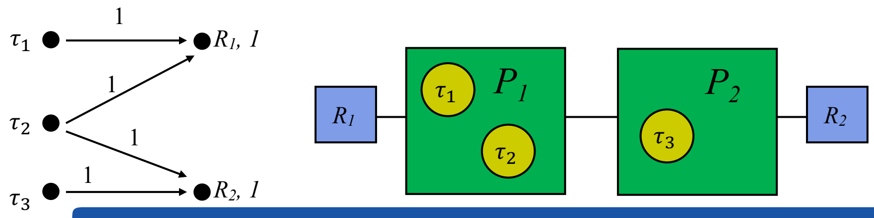


66

Local vs. Remote Resources

End-to-End Resource Model

- Since τ_1 executes only on P_1 and τ_3 executes only on P_2 they consist only of one component
- Using this definition, each component job requires only resources on the processor on which the component jobs executes

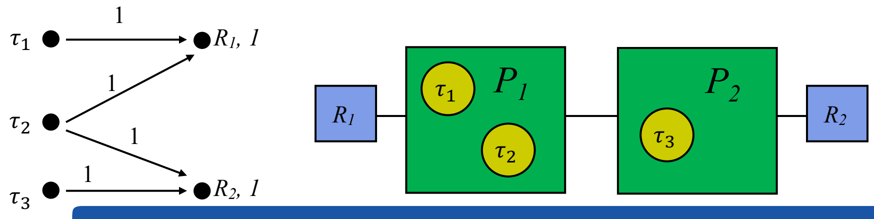


67

Local vs. Remote Resources

End-to-End Resource Model

- The scheduler of each processor can treat all requests for resources controlled by it as local requests
- No need to distinguish tasks from component tasks
- All tasks in an end-to-end task requires only resources on its local processor



68

Multiprocessor Priority-Ceiling Protocol (MPCP)

Assumption

- Tasks and resources have been assigned and statically bound to processors
- Scheduler of every synchronization processor knows the priorities and resource requirements of all tasks requiring the global resources managed by the processor

69

Multiprocessor Priority-Ceiling Protocol (MPCP)

- The scheduler of each processor schedules all the local tasks and global critical sections on a processor on a fixed-priority basis
- Resource accesses are controlled by a modified priority-ceiling protocol

Multiprocessor Priority-Ceiling Protocol (MPCP)

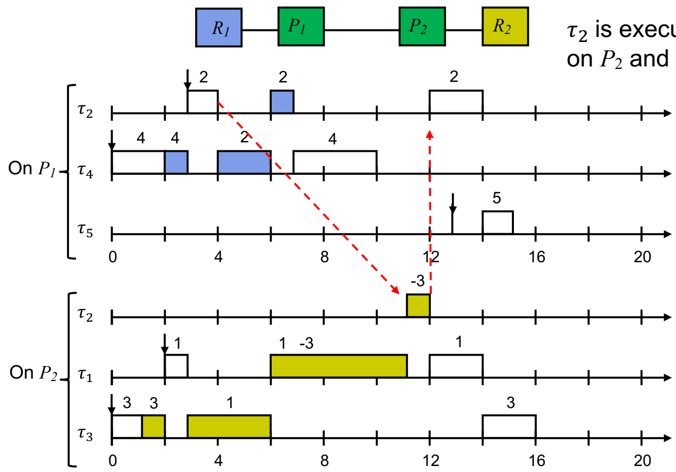
- The multiprocessor priority-ceiling protocol schedules all global critical sections at *higher* priorities than all local tasks on every synchronization processor
- Idea is that local tasks shall not delay execution of global critical sections and prolong the blocking time of the remote task

Multiprocessor Priority-Ceiling Protocol (MPCP)

- This is implemented if the lowest priority π_{lowest} of all tasks in a system is known
- Then the global critical sections of a task with priority π_i are executed at a priority π_i - π_{lowest}

72

Multiprocessor Priority-Ceiling Protocol (MPCP)



τ_2 is executed first on P_1 then on P_2 and then again on P_1

73

Multiprocessor Priority-Ceiling Protocol (MPCP)

The blocking time $B_i(rc)$ for a task consist of the following contributions

1. *local blocking time*: caused by contentions on the local processor
2. *local preemption delay*: caused by global critical sections that belong to remote tasks executing on the local processor
3. *remote blocking time*: caused by contention with lower-priority tasks that execute on the remote processor

Multiprocessor Priority-Ceiling Protocol (MPCP)

The blocking time $B_i(rc)$ for a task consist of the following contributions

4. *remote preemption delay*: caused by preemptions of other higher-priority remote sections executing on the same remote processor
5. *deferred blocking time*: suspended execution of local higher-priority tasks

Formal Model in Liu, Section 9.3.2 – not discussed in this course

Summary

The theory for uniprocessor forms the foundation for static multiprocessor systems

New problems arise

- Task Assignment
- Multiprocessor protocols
- Timing Anomalies

Naturally, the problems get much more complex

76

References

Davis and Burns, A Survey of Hard Real-Time Scheduling for Multiprocessor Systems, *ACM Computing Surveys*, Vol. 43, No. 4, October 2011.

77