CS4272: HW SW Codesign

Scheduling

Dr. Abhik Roychoudhury
School of Computing
National University of Singapore

Some slides are modified from Peter Marwedel's accompanying lecture notes

Hardware Software partioning

- Deciding which parts of the design will be implemented where --- on processor or as custom hardware
  - Allocation of tasks to processing elements
  - More than two tasks may get allocated to the same processing element
  - If so, how will they share the computing power of the processing element
    - Scheduling methods --- today's lecture !

Reading

- Section 4.1, 4.2 of textbook
  - Embedded System Design
  - Peter Marwedel
  - Coverage is quite good
- Word of caution
  - There are many scheduling algorithms for different settings
  - Our aim is to understand the basic concepts, not to give a list of all scheduling algorithms.

Organization

- Real-time Systems
- Basics of Scheduling
- Aperiodic Scheduling Methods
- Periodic Scheduling Methods
  - RMS
  - EDF
- Resource Access Protocols

Definition

- Embedded systems that monitor, respond to, or control external environment under real time constraints
- Examples:
  - Vehicles (car, aircraft, …)
  - Traffic control (highway, air, railway, …)
  - Process control (power plant, chemical plant, …)
  - Medical systems (radiation therapy, …)
  - Telephone, radio, satellite communication
  - Computer games

Characteristics

- Timing constraints / deadline
  - Functional and temporal correctness
- Hard deadline
  - Must always meet deadline
  - Air traffic controller
- Soft deadline
  - Must frequently meet deadline
  - MPEG decoder

Characteristics (Contd.)

- Concurrency (multiple processes)
  - Handle multiple input and output signals
- Reliability
  - How often the system fails
- Fault tolerance
  - Recognition and handling of failures
- Critical system
  - High cost of failure
  - Hard real time system $\Rightarrow$ critical system

Tasks

- A task is a block of code executed in a CPU in a sequential fashion.
- Several independent tasks may be executing on the same CPU
  - How to schedule them ?
  - Today's lecture
- There might also be dependences among tasks, captured by a task graph
  - Task mapping – which task on which CPU?
  - Task scheduling – in what order to run the tasks mapped to same CPU?

Why study scheduling?

- Increase CPU utilization or other metrics
- For real-time systems requiring hard guarantees
  - Study in advance whether all tasks can be scheduled without missing any deadlines.
  - Need computation time of each task
    - Typically given as a worst-case bound, called the Worst-case Execution Time (WCET)
    - How to derive these bounds ?
    - Involves a low-level code analysis, will be discussed in later lectures.

Organization

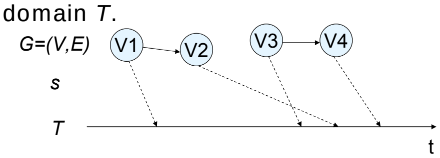
- Real-time Systems
- Basics of Scheduling
- Aperiodic Scheduling Methods
- Periodic Scheduling Methods
  - RMS
  - EDF
- Resource Access Protocols

Informally, scheduling is

- Assume that we are given a task graph $G=(V,E)$.
- **Def.:** A **schedule** s of G is a mapping

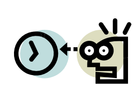
$$V \rightarrow T$$

of a set of tasks V to start times from domain T.



Informally, scheduling is

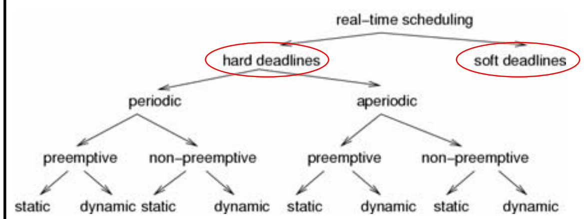
- Typically, schedules have to respect a number of constraints, incl. resource constraints, dependency constraints, deadlines.
- **Scheduling** = finding such a mapping.
- Scheduling to be performed several times during ES design (early rough scheduling as well as late precise scheduling).



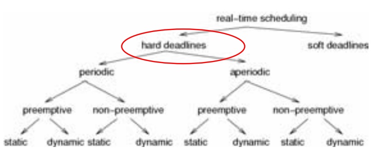
More precisely,

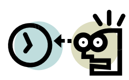
- Schedule
  - An assignment of tasks to the processor (assuming 1 processor!) over time.
- Feasible schedule
  - All tasks can be completed and all constraints (precedence, resource, deadline) can be respected.
- Scheduling Algorithm
  - A recipe for producing schedules
- Schedulability
  - If at least one scheduling algorithm producing a feasible schedule exists.

Classification of scheduling algorithms



Hard and soft deadlines

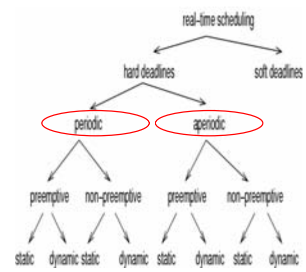




- **Def.:** A time-constraint (deadline) is called **hard** if not meeting that constraint could result in a catastrophe [Kopetz, 1997].
- All other time constraints are called **sof**t.

- We will focus on hard deadlines.

Periodic and aperiodic tasks



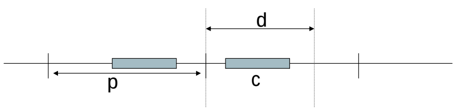
Periodic and Aperiodic

- **Def.:** Tasks which must be executed once every *p* units of time are called **periodic** tasks. *p* is called their period. Each execution of a periodic task is called a **job**.
- All other tasks are called **aperiodic**.
- **Def.:** Tasks requesting the processor at unpredictable times are called **sporadic**, if there is a minimum separation between the times at which they request the processor.

Periodic Task

- Activated on a regular basis between fixed interval
  - scan the airspace every 3 sec
- P = (s, c, p, d)
  - s = start time or arrival time
  - c = worst case execution time (WCET)
  - p = period or cycle time
  - d = deadline
  - c <= d <= p

Periodic Process (Contd.)



- **Period**: interval between process activations.
- **Initiation interval**: reciprocal of period.
- **Initiation time**: time at which process becomes ready.
- **Deadline**: time at which process must finish.
- In most cases, d = p

Sporadic Task

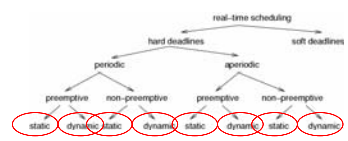
- P = (c, p, d) where c <= d <= p
- $t \leq t_e + d$
  where t is completion time
  t_e is the event occurrence time
- p is the minimum time between event
- If p = 0, then aperiodic task
- Aperiodic task does not have deterministic timing constraints
- Jitter : Variation from cycle to cycle in task completion time

Preemptive and non-preemptive scheduling

- **Non-preemptive schedulers:**
  Tasks are executed until they are done.
  Response time for external events may be quite long.
- **Preemptive schedulers:** To be used if
  - some tasks have long execution times or
  - if the response time for external events to be short.

Dynamic/online scheduling

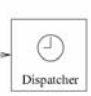
- **Dynamic/online scheduling:**
  Processor allocation decisions (scheduling) at run-time; based on the information about the tasks arrived so far.



Static/offline scheduling

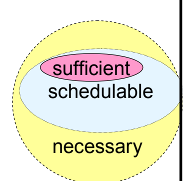
- **Static/offline scheduling:**
  Scheduling taking a priori knowledge about arrival times, execution times, and deadlines into account.
  Dispatcher allocates processor when interrupted by timer. Timer controlled by a table generated at design time.

Time	Action	WCET
10	start T1	12
17	send M5	
22	stop T1	
38	start T2	20
47	send M3	



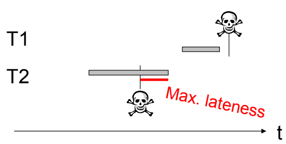
Schedulability

- Set of tasks is **schedulable** under a set of constraints, if a schedule exists for that set of tasks & constraints.
- **Exact tests** are NP-hard in many situations.
- **Sufficient tests**: sufficient conditions for schedule checked. (Hopefully) small probability of indicating that no schedule exists even though one exists.
- **Necessary tests**: checking necessary conditions. Used to show no schedule exists. There may be cases in which no schedule exists & we cannot prove it.



Cost functions

- **Cost function:** Different algorithms aim at minimizing different functions.
- **Def.: Maximum lateness =**
- $\max_{\text{all tasks}}$ (completion time – deadline)
  Is <0 if all tasks complete before deadline.

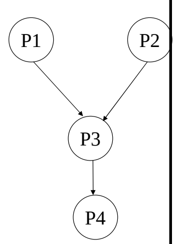


To summarize

- Input to Scheduling Algorithm
  - One or more processes
  - Activation time, execution time, deadline for each process
- Scheduling algorithm: a policy to allocate tasks to the processor(s)
- Feasible schedule if the scheduling algorithm can meet all the constraints
- Optimal algorithm: A scheduling algorithm that produces a feasible schedule if it exists

To summarize

- Constraints to be met by scheduling algorithm
  - No task misses deadlines
  - Precedence constraints among tasks --- captured by task graph
  - Resource constraints
    - Due to synchronization over shared data structures/resources by different tasks
    - We will discuss this later.



To summarize

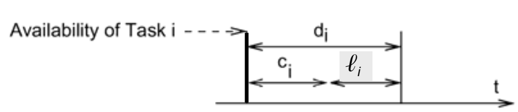
- How do we evaluate a scheduling policy:
  - Ability to satisfy all deadlines.
  - CPU utilization: percentage of time devoted to useful work.
  - Scheduling overhead: time required to make scheduling decision.

Organization

- Real-time Systems
- Basics of Scheduling
- Aperiodic Scheduling Methods
- Periodic Scheduling Methods
  - RMS
  - EDF
- Resource Access Protocols

Aperiodic scheduling - Scheduling with no precedence constraints

- Let $\{T_i\}$ be a set of tasks. Let:
- c_i be the execution time of T_i ,
- d_i be the **deadline interval**, that is, the time between T_i becoming available and the time until which T_i has to finish execution.
- ℓ_i be the **laxity** or **slack**, defined as $\ell_i = d_i - c_i$
- f_i be the finishing time.

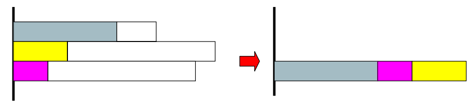


Our very first …

- … scheduling problem
  - Uni-processor
  - Set of independent tasks
  - All tasks arrive at the same time.

Uni-processor with equal arrival times

- Preemption is useless.
- **Earliest Due Date** (EDD): Execute task with earliest due date (deadline) first.



- EDD requires all tasks to be sorted by their (absolute) deadlines. Hence, its complexity is $O(n \log(n))$.

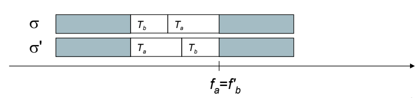
More in-depth:

Optimality of EDD

- EDD is optimal, since it follows Jackson's rule: Given a set of n independent tasks, any algorithm that executes the tasks in order of non-decreasing (absolute) deadlines is optimal with respect to minimizing the maximum lateness.
- Proof (See Buttazzo, 2002):
  - Let σ be a schedule produced by any algorithm A
  - If $A \neq$ EDD $\rightarrow \exists\ T_a, T_b, d_a \leq d_b, T_b$ immediately precedes T_a in σ.
  - Let σ' be the schedule obtained by exchanging T_a and T_b.

Exchanging T_a and T_b cannot increase lateness

- Max. lateness for T_a and T_b in σ is $L_{max}(a,b)=f_a-d_a$
- Max. lateness for T_a and T_b in σ' is $L'_{max}(a,b)=\max(L'_a,L'_b)$
- Two possible cases
  1. $L'_a \geq L'_b$: $\rightarrow L'_{max}(a,b) = f'_a - d_a < f_a - d_a = L_{max}(a,b)$ since T_a starts earlier in schedule σ'.
  2. $L'_a \leq L'_b$: $\rightarrow L'_{max}(a,b) = f'_b - d_b = f_a - d_b \leq f_a - d_a = L_{max}(a,b)$ since $f_a=f'_b$ and $d_a \leq d_b$
- ☞ $L'_{max}(a,b) \leq L_{max}(a,b)$



EDD is optimal

end

☞ Any schedule σ with lateness L can be transformed into an EDD schedule σ^n with lateness $L^n \leq L$, which is the minimum lateness.

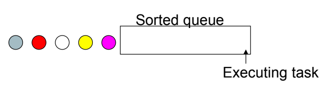
☞ EDD is optimal (q.e.d.)

Earliest Deadline First (EDF) - Horn's Theorem -

- Different arrival times: Preemption potentially reduces lateness.
- **Theorem** [Horn74]: Given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

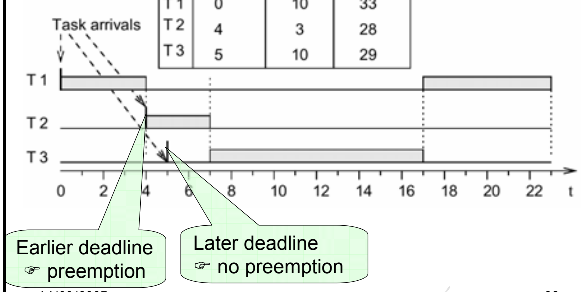
Earliest Deadline First (EDF) - Algorithm -

- **Earliest deadline first** (EDF) algorithm:
  - Each time a new ready task arrives:
  - It is inserted into a queue of ready tasks, sorted by their **absolute** deadlines. Task at head of queue is executed.
  - If a newly arrived task is inserted at the head of the queue, the currently executing task is preempted.
- Straightforward approach with sorted lists (full comparison with existing tasks for each arriving task) requires run-time $O(n^2)$



Earliest Deadline First (EDF) - Example -

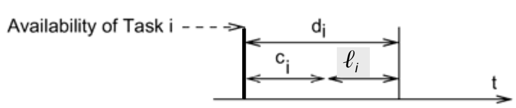
	arrival	duration	deadline
T1	0	10	33
T2	4	3	28
T3	5	10	29



Another Algorithm

Least Laxity first

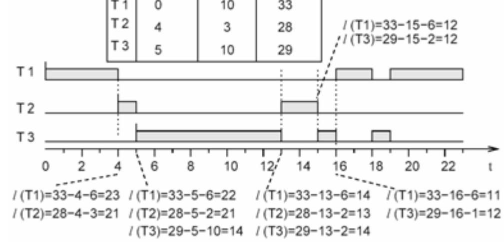
--- Another dynamic priority algorithm, but different from EDF.



Least laxity (LL), Least Slack Time First (LST)

- Priorities = decreasing function of the laxity (the less laxity, the higher the priority); dynamically changing priority; preemptive.

	arrival	duration	deadline
T1	0	10	33
T2	4	3	28
T3	5	10	29

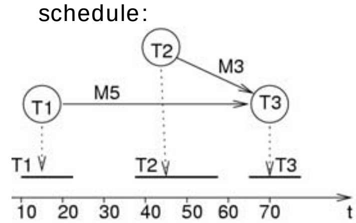


Summarizing ...

- Both EDF and LL are optimal for
  - Uni-procesor
  - Aperiodic independent tasks
  - Arrival times diff, pre-emption allowed
  - No precedence, resource or other constraints
- This means that both algorithms will find a schedule, if one exists --- but ...
  - Their produced schedules may be different.

Scheduling with precedence constraints

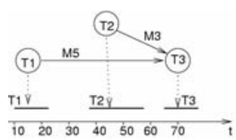
- Task graph and possible schedule:



Schedule can be stored in table.

Simultaneous Arrival Times: The Latest Deadline First (LDF) Algorithm

- LDF [Lawler, 1973]: reads the task graph and among the tasks with no successors inserts the one with the latest deadline into a queue. It then repeats this process, putting tasks whose successor have all been selected into the queue.
- At run-time, the tasks are executed in the generated total order.
- LDF is non-preemptive and is optimal for uni-processors.



If no local deadlines exist, LDF performs just a topological sort.

What if

- We have …
  - Uni-processor
  - Aperiodic Tasks with precedence constraints
  - Diff arrival times and pre-emption
- We have …
  - Uni-processor
  - Aperiodic Tasks with precedence constraints
  - Diff arrival times and no pre-emption
- We have …
  - Omitted from our "laundry list" ☺

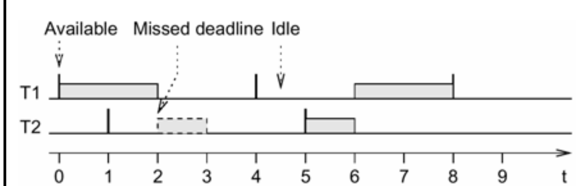
Scheduling without preemption

Lemma: If preemption is not allowed, optimal schedules may have to leave the processor idle at certain times.

Proof: Suppose: optimal schedulers never leave processor idle.

Scheduling without preemption

- T1: periodic, $c_1 = 2$, $p_1 = 4$, $d_1 = 4$
- T2: occasionally available at times 4*n+1, $c_2 = 1$, $d_2 = 1$
- T1 has to start at t=0
- ☞ deadline missed, but schedule is possible (start T2 first)
- ☞ scheduler is not optimal ☞ contradiction! q.e.d.



Scheduling without preemption

- Preemption not allowed: ☞ optimal schedules may leave processor idle to finish tasks with early deadlines arriving late.

☞Knowledge about the future is needed for optimal scheduling algorithms then

☞☞No online algorithm can decide whether or not to keep idle.

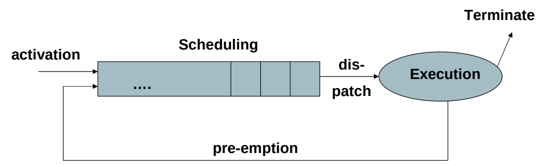
Organization

- Real-time Systems
- Basics of Scheduling
- Aperiodic Scheduling Methods
- Periodic Scheduling Methods
  - RMS
  - EDF
- Resource Access Protocols

Periodic task - Recap

- Activated on a regular basis between fixed interval
  - scan the airspace every 3 sec
- P = (s, c, p, d)
  - s = start time or arrival time
  - c = worst case execution time (WCET)
  - p = period or cycle time
  - d = deadline
  - c <= d <= p

Task Execution Recap



Dispatching from the ready queue will be based on the scheduling policy which takes into account task priority.

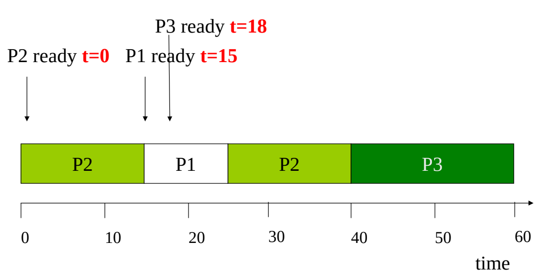
Priority-driven scheduling

- Each process has a priority.
- CPU goes to highest-priority process that is ready.
- Priorities determine scheduling policy:
  - fixed priority;
  - time-varying priorities.

Priority-driven scheduling example

- Rules:
  - each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU
    - Preemptive scheduling
  - process continues until done.
- Processes
  - P1: priority 1, execution time 10
  - P2: priority 2, execution time 30
  - P3: priority 3, execution time 20

Priority-driven scheduling example



Rate monotonic scheduling

- RMS (Liu and Layland 1973)
  - widely-used, analyzable scheduling policy.
- Analysis is known as Rate Monotonic Analysis
- RMS is an optimal fixed priority assignment method
  - If there exists a schedule that meets all the deadlines with fixed priority, then RMS will produce a feasible schedule

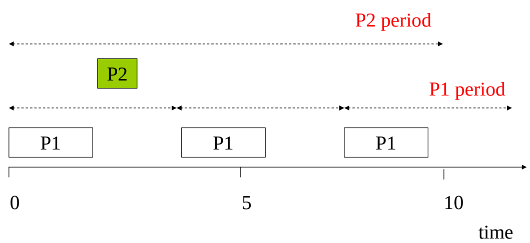
RMA model

- All process run on single CPU.
- Zero context switch time.
- No data dependencies between processes.
- Process execution time is constant.
- Deadline is at end of period (p = d)
- Highest-priority ready process runs.

RMS priorities

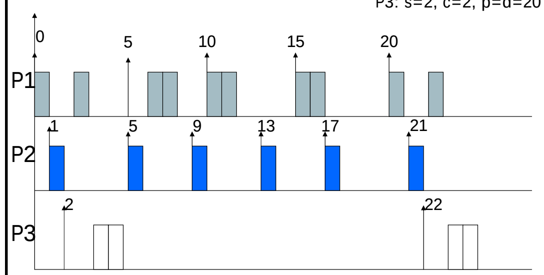
- Optimal (fixed) priority assignment:
  - shortest-period process gets highest priority;
  - priority inversely proportional to period;
  - break ties arbitrarily.
- Intuition: Processes requiring frequent attention (smaller period) should receive higher priority

RMS example



RMS Example

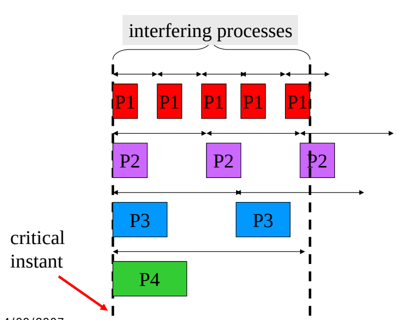
P1: s=0, c=2, p=d=5
P2: s=1, c=1, p=d=4
P3: s=2, c=2, p=d=20



Rate-monotonic analysis

- Response time: time required to finish process.
- Critical instant: scheduling state that gives worst response time.
- Critical instant occurs when all higher-priority processes are ready to execute.

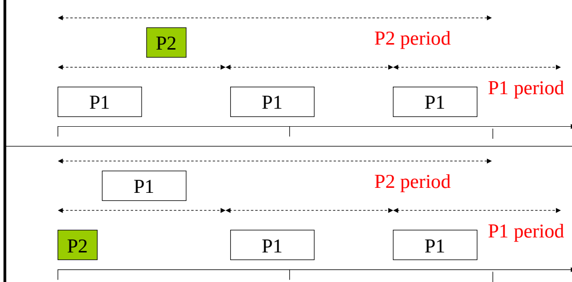
Critical instant



Informal argument of optimality

- P1 = (c1, p1, d1) with p1 = d1
- P2 = (c2, p2, d2) with p2 = d2
- p1 < p2
- Suppose P1 and P2 can be scheduled with non-RM priority assignment, i.e., P2 has highest priority
- At critical instant, with non-RM priorities
  - c1 + c2 <= p1; [1]
- With RM priority
  - ⌊p2/p1⌋*c1 + c2 <= p2; [2]
- If [1] is satisfied, then [2] is also satisfied

RMS optimality



RMS CPU utilization

- Utilization for n processes is
  - $U = \Sigma_i \, c_i / p_i$
- U<=1 is a necessary condition for feasibility regardless of scheduling policy
- Scheduling with fixed priorities is feasible if
  $U <= n (2^{1/n} – 1)$
- The bound is sufficient but not necessary
- As number of tasks approaches infinity, maximum utilization approaches 69%.
  - RMS cannot use 100% of CPU, even with zero context switch overhead.
  - Must keep idle cycles available to handle worst-case scenario.

Practical side

- RMS is widely used.
  - Static priority scheme makes it easy to implemented.
  - Implemented within OS to manage scheduling of threads
    - Windows NT

Earliest-deadline-first scheduling

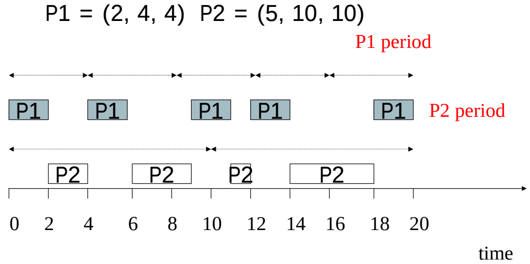
- EDF: dynamic priority scheduling scheme.
- Process closest to its deadline has highest priority.
- Requires recalculating processes at every timer interrupt.
- EDF can use 100% of CPU.

EDF implementation

- On each timer interrupt:
  - compute time to deadline;
  - choose process closest to deadline.
- Generally considered too expensive to use in practice.

EDF example

P1 = (2, 4, 4) P2 = (5, 10, 10)



EDF Results

- EDF is optimal
  - If a feasible schedule exists using dynamic priorities, then EDF will produce a feasible schedule
- EDF can always produce a feasible schedule if $U \leq 1$
- Scheduling with dynamic priority is feasible if and only if $U \leq 1$

Fixing scheduling problems

- What if your set of processes is not schedulable?
  - Change deadlines in requirements.
  - Reduce execution times of processes.
  - Get a faster CPU.

Organization

- Real-time Systems
- Basics of Scheduling
- Aperiodic Scheduling Methods
- Periodic Scheduling Methods
  - RMS
  - EDF
- Resource Access Protocols

Resource Constraints

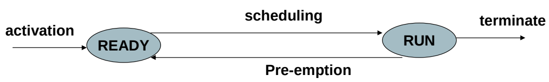
- Resource:
  - software structure used by a task during its execution.
  - A data structure, variables, an area of main memory, a file, a set of registers of a peripheral device.
- Shared resource:
  - Used by more than one task.
- Exclusive resource:
  - No simultaneous access.
  - Require mutual exclusion.
  - Operating system must provide a synchronization mechanism to ensure sequential access..

A small question

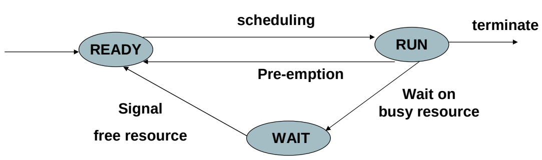
- If the tasks are Java threads
  - Of a program you have written
- The scheduler is in the OS/VM
  - Depends on the VM
    - Sun's VM uses scheduler in OS
    - Kaffe VM manages its own scheduling
- What kind of exclusive shared resources can you think of?

Task states

Without Resource Constraints

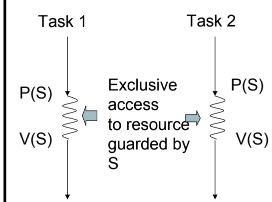


With Resource Constraints



Resource access protocols

- **Critical sections:** sections of code at which exclusive access to some resource must be guaranteed.
- Can be guaranteed with semaphores S.

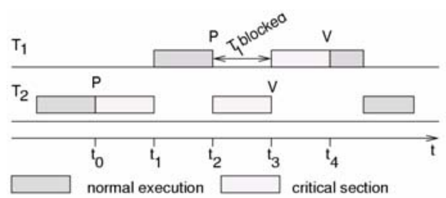


P(S) checks semaphore to see if resource is available and if yes, sets S to "used". Uninterruptable operations! If no, calling task has to wait.

V(S): sets S to "unused" and starts sleeping task (if any).

Priority inversion

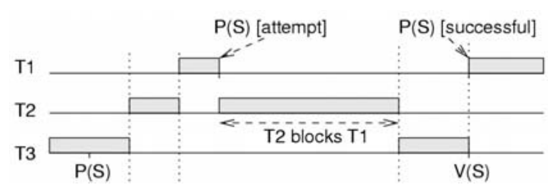
- Priority T_1 assumed to be > than priority of T_2.
- If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (time t_3), thus inverting the priority:



In this example:
duration of inversion bounded by length of critical section of T_2.

Duration of priority inversion with >2 tasks can exceed the length of any critical section

- Priority of T1 > priority of T2 > priority of T3.
- T2 preempts T3:
- T2 can prevent T3 from releasing the resource.



The MARS Pathfinder problem

- "But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once"." …

The MARS Pathfinder problem

- "VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks."
- "Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft."
  - A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes)."

The MARS Pathfinder problem

- The meteorological data gathering task ran as an infrequent, low priority thread, … When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority."

☞
High priority: retrieval of data from shared memory
Medium priority: communications task
Low priority: thread collecting meteorological data

The MARS Pathfinder problem (4)

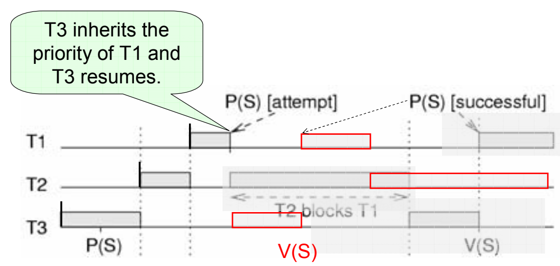
➢"Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset. This scenario is a classic case of priority inversion."

Coping with priority inversion: the priority inheritance protocol

- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- If task T1 executes P(S) & exclusive access granted to T2: T1 will become blocked.
  If priority(T2) < priority(T1): T2 inherits the priority of T1.
  ☞ T2 resumes.
  Rule: tasks inherit the highest priority of tasks blocked by it.
- When T2 executes V(S), its priority is decreased to the highest priority of the tasks blocked by it.
  If no other task blocked by T2: priority(T2):= original value.
  Highest priority task so far blocked on S is resumed.
- Transitive: if T2 blocks T1 and T1 blocks T0, then T2 inherits the priority of T0.

Example

➢How would priority inheritance affect our example with 3 tasks?



Priority inversion on Mars

- ➢ Priority inheritance also solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements a flag for the calls to mutex primitives. This flag allows priority inheritance to be set to "on". When the software was shipped, it was set to "off".

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to "on", while the Pathfinder was already on the Mars [Jones, 1997].

Remarks on priority inheritance protocol

➢Possible large number of tasks with high priority.

➢Possible deadlocks.

➢Ongoing debate about problems with the protocol:

Victor Yodaiken: Against Priority Inheritance, http://www.fsmlabs.com/articles/inherit/inherit.html

➢Finds application in ADA: During *rendez-vous*, task priority is set to the maximum.

➢More sophisticated protocol: priority ceiling protocol.

Deadlocks on inheritance

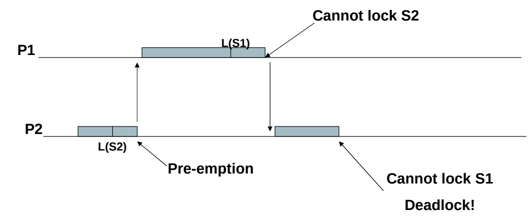
Two tasks with nested critical sections

Task P1
```
...
  lock(S1)
  ...
      lock(S2)
      ...
      unlock(S2)
  ...
  unlock(S1)
  ...
```

Task P2
```
...
  lock(S2)
  ...
      lock(S1)
      ...
      unlock(S1)
   ...
  unlock(S2)
  ...
```

14/09/2007

Deadlock Example



Assume that P1 has higher priority than P2

14/09/2007

Priority Ceiling Protocol

- Basic Idea:
  - A task is not allowed to enter a critical section if there are already locked semaphores which could block it eventually.
  - Once a task enters a critical section, it cannot be blocked by lower priority tasks till its completion.

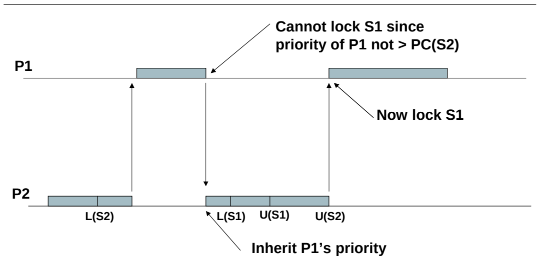
14/09/2007

Priority Ceiling Protocol

- Define Priority ceiling of a resource
  - PC(S) = highest priority of all tasks that may lock S
  - A task T attempting to lock a resource will be suspended unless its priority is higher than PC(S) for all resources S currently locked by all tasks other than T.
    - This means --- if any of the currently locked resources can be used by T, it is suspended.
  - If T is suspended then the task T1 that holds the lock with highest PC, effectively blocks T
    - T1 inherits T's priority as in priority inheritance protocol.

14/09/2007

No deadlock



PC(S1) = PC(S2) = max(P1's priority, P2's priority) = P1's priority

14/09/2007

More Explanation (1)

- Critical Section Entry
  - Let S* be the semaphore with the highest priority ceiling among all the semaphores currently locked by tasks other than T.
  - To enter the critical section guarded by S , T's priority must be higher than PC(S*).
  - If not, the lock on S is denied.
    - T is now said to be blocked on semaphore S*.
  - When T is thus blocked it transmits its priority to the task, say T*, that is holding the semaphore S* which is blocking T.
    - T* will now start executing.

14/09/2007

More Explanation (2)

- Critical Section Exit
  - Similar to Priority Inheritance protocol
    - When the currently executing T* exits a critical section, it unlocks the semaphore, and the highest priority task, if any, that is blocked on that semaphore is awakened.
    - The priority of T* is set to the priority of the highest priority task it is continuing to block.

Wrapping up

- Schedulability analysis
  - Aperiodic and periodic
  - All tasks arrive at same time --- simplistic
    - Or at diff. times – do we allow pre-emptions
  - Popular methods for periodic tasks
    - RMS (static priority), EDF (dynamic priority)
    - Static priority of RMS makes it easy to implement.
  - Resource access protocols
    - Tasks may share resources
    - Priority inheritence and priority ceiling protocols.

RMS & EDF classroom exercise

	t_1	t_2	t_3
C_i	1	2	3
p_i	4	6	10

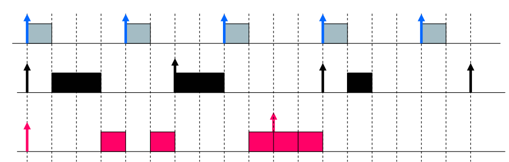
All tasks are periodic, period = deadline

All tasks arrive at time 0

Construct a RMS schedule.

RMS schedule

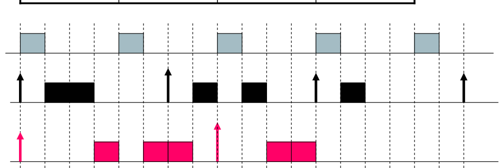
	t_1	t_2	t_3
C_i	1	2	3
p_i	4	6	10



Construct EDF schedule

All tasks arrive at t=0

	t_1	t_2	t_3
C_i	1	2	3
p_i	4	6	8



More Classroom Exercises

- Consider the following scheduling algorithm for periodic tasks $P_i = (c_i, p_i, p_i)$, where the execution time is c_i, the period is p_i and the deadline is also p_i. Assume that all execution times c_i and periods p_i are given by integers, for simplicity. The scheduling algorithm proceeds as follows. In every unit of time, we allocate a fraction of the CPU equal to c_i / p_i to each task P_i Show any one run of this scheduling algorithm on the following three tasks
- P1 = (1,4,4), P2 = (2, 6, 6), P3 = (1,3,3).

Answers

In every time unit, the three processes get allocated the following fractions of time (in any order). Shown for the first 6 time units.

	P1	P2	P3	
Time unit 1	¼	1/3	1/3	
2	¼	1/3	1/3	
3	¼	1/3	1/3	P3 finished meets deadline
4	¼	1/3	1/3	P1 finished meets deadline
5	¼	1/3	1/3	
6	¼	1/3	1/3	P2 finished meets deadline

More Classroom Exercises

- Let us call the scheduling algorithm in Question 2A as *OurSchedAlgo*. Can you compare it with RMS and EDF? That is, whenever RMS produces a feasible schedule will *OurSchedAlgo* produce a feasible schedule and vice-versa? Similarly, whenever EDF (with preemption) produces a feasible schedule will *OurSchedAlgo* produce a feasible schedule and vice-versa? Give detailed justifications for your answer.

Answers

The algorithm is optimal, i.e. whenever a feasible schedule exists, it will find one.
In every unit of time, process i gets ci /pi units of time.

So, in pi units of time, process i will get ci units of time and thus meet its deadline, unless the summation of ci/pi for all i (the utilization) is greater than 1.

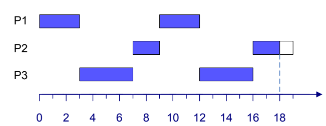
This means that OurSchedAlgo has the same feasibility region as EDF.

Furthermore, OurSchedAlgo should have a larger feasibility region than RMS --- all problems which cannot be scheduled using static priorities/RMS, but can be scheduled using dynamic priorities/EDF are examples where OurSchedAlgo will produce a schedule, but RMS will not.

One such example is given in next slide

Answers

P1 = (3,9,9), P2 = (5,18,18), P3 = (4,12,12).



Using RMS, P2 is not finished (1 time unit left) when its deadline is reached at 18
Since the utilization factor is 3/9 + 5/18 + 4/12 = 0.944 < 1, OurSchedAlgo is guaranteed to produce a schedule.

Midterm Examination

- **11th Oct** Thu 9 AM, SR 3B
  - Basics, Modeling, StateCharts, Scheduling, Partitioning
  - Open Book --- bring in any material.
  - 2 hour exam.