



Administrative matter


- Quiz 2 (30 minutes)
 - Cover ESD chapter 10,11, labs till Trimedia lab
- Homework #6 posted
 - Due Tuesday 12/5
- No lecture on 11/28
 - Happy Thanksgiving
- Final examination
 - Tuesday 12/10, 8-11AM
 - In classroom
 - Cover ESD 8,10,11
 - Cover RTSPL 5, 13.1-13.13
 - Cover all labs




CS122A: Embedded System Design, Fall01
1


Process-Based Scheduling

- Scheduling approaches
 - Fixed-Priority Scheduling (FPS)
 - Earliest Deadline First (EDF)
 - Value-Based Scheduling (VBS)


CS122A: Embedded System Design, Fall01
2


Fixed-Priority Scheduling (FPS)

- This is the most widely used approach
- Each process has a fixed, static, priority, computed pre-run-time
- Runnable processes are executed in order determined by priority
- “priority” of a process is derived from its temporal requirements
 - not its importance to the correct functioning of the system or its integrity


CS122A: Embedded System Design, Fall01
3


Earliest Deadline First (EDF) Scheduling

- The runnable processes are executed in the order
 - determined by the absolute deadlines of the processes
- The next process to run
 - being the one with the shortest (nearest) deadline
- It is usual to know the relative deadlines of each process
 - e.g. 25ms after release
- Absolute deadlines are computed at run time
 - hence the scheme is described as dynamic


CS122A: Embedded System Design, Fall01
4


Value-Based Scheduling (VBS)

- If a system can become overloaded then the use of simple static priorities or deadlines is not sufficient; a more adaptive scheme is needed
- This often takes the form of assigning a value to each process and employing an on-line value-based scheduling algorithm to decide which process to run next
- Will be covered later if time permits...


CS122A: Embedded System Design, Fall01
5

Preemption and Non-preemption

- A high-priority process may be released during the execution of a lower priority one
- preemptive schemes
 - an immediate switch to the higher-priority process
- non-preemption schemes,
 - the lower-priority process will complete before the other executes
- Preemption enable higher-priority processes to be more reactive
- Deferred preemption or cooperative dispatching
 - strategies allow a lower priority process to continue for bounded time
- Schemes such as EDF and VBS can be either


CS122A: Embedded System Design, Fall01
6

FPS and Rate Monotonic Priority Assignment

- Each process is assigned a unique priority based on its period
- The shorter the period, the higher the priority
- I.e, for two processes i and j,

$$T_i < T_j \Rightarrow P_i > P_j$$
- This assignment is optimal
- if any process set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme
 - then the given process set can also be scheduled with a rate monotonic assignment scheme

Note, priority 1 is the lowest (least) priority

CS122A: Embedded System Design, Fall01 7

Example Priority Assignment

Process	Period, T	Priority, P
a	25	5
b	60	3
c	42	4
d	105	1
e	75	2

CS122A: Embedded System Design, Fall01 8

Utilization-Based Analysis

- For D=T task sets only
- A simple sufficient but not necessary schedulability test exists

$$U \equiv \sum_{i=1}^N \frac{C_i}{T_i} \leq N (2^{1/N} - 1)$$

$U \leq 0.69$ as $N \rightarrow \infty$

CS122A: Embedded System Design, Fall01 9

Utilization Bounds

N	Utilization bound
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

Approaches 69.3% asymptotically

CS122A: Embedded System Design, Fall01 10

Process Set A

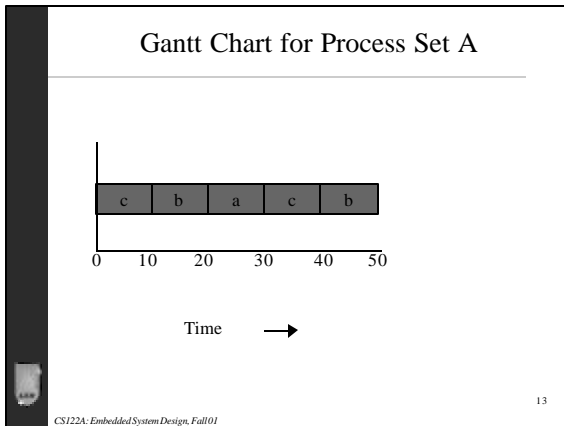
Process	Period T	ComputationTime C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33

- The combined utilization is 0.82 (or 82%)
- This is above the threshold for three processes (0.78)
 - hence, this process set fails the utilization test

CS122A: Embedded System Design, Fall01 11

Time-line for Process Set A

CS122A: Embedded System Design, Fall01 12



Process Set B

Process	Period T	ComputationTime C	Priority P	Utilization U
a	80	32	1	0.400
b	40	5	2	0.125
c	16	4	3	0.250

- The combined utilization is 0.775
- This is below the threshold for three processes (0.78)
 - hence, this process set will meet all its deadlines

14

CS122A: Embedded System Design, Fall 01

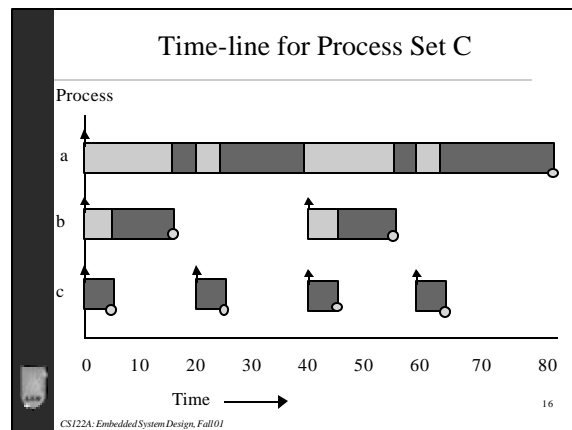
Process Set C

Process	Period T	ComputationTime C	Priority P	Utilization U
a	80	40	1	0.50
b	40	10	2	0.25
c	20	5	3	0.25

- The combined utilization is 1.0
- This is above the threshold for three processes (0.78)
 - but the process set will meet all its deadlines

15

CS122A: Embedded System Design, Fall 01



Criticism of Utilization-based Tests

- Not exact
- Not general
- BUT it is O(N)

The test is said to be sufficient but not necessary

17

CS122A: Embedded System Design, Fall 01

Utilization-based Test for EDF

A much simpler test

$$\sum_{i=1}^N \frac{C_i}{T_i} \leq 1$$

- High utilizations than FPS
- Harder to implement than FPS
- Requires a more complex run-time system, I.e. higher overhead
- Processes without deadline
 - FPS: Just set deadline to infinite, and give it arbitrary (low) priority
 - EDF: some arbitrary large deadline, a little more awkward
- During overload situations
 - FPS is more predictable;
 - Low priority process miss their deadlines first
 - EDF is unpredictable
 - domino effect can occur in which many processes miss deadlines

18

CS122A: Embedded System Design, Fall 01

Earliest Deadline First

- At any given time, task with the earliest deadline goes
 - Obviously, very high overhead
 - Have to keep checking deadline
- Dynamic Priority
- Feasible if processor usage < 1
- For task running 5ms (usage .9585)
 - Assume deadline=period

Process	Period	Priority
A	25 ms	5
B	50 ms	3
C	12 ms	6
D	100 ms	1
E	40 ms	4
F	75 ms	2

19

Response-Time Analysis

- Task i 's worst-case response time, R , is calculated first and then checked (trivially) with its deadline

$$R_i \leq D_i$$

$$R_i = C_i + I_i$$

Where I is the interference from higher priority tasks

20

Calculating R

During R , each higher priority task j will execute a number of times:

$$\text{Number of Releases} = \left\lceil \frac{R_i}{T_j} \right\rceil$$

The ceiling function gives the smallest integer greater than the fractional number on which it acts. So the ceiling of $1/3$ is 1, of $6/5$ is 2, and of $6/3$ is 2.

Total interference is given by:

$$\sum \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

21

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

The set of values $w_i^0, w_i^1, w_i^2, \dots, w_i^n$ is monotonically non decreasing. When $w_i^n = w_i^{n+1}$ the solution to the equation has been found, w_i^0 must not be greater than w_i^n (e.g. 0 or ∞)

22

Response Time Algorithm

```

for i in 1..N loop -- for each process in turn
  n := 0
  loop
    calculate new w_i^{n+1}
    if w_i^{n+1} < w_i^n then
      exit value found
    end if
    if w_i^{n+1} > w_i^n then
      exit value not found
    end if
    n := n + 1
  end loop
end loop
    
```

23

Process Set D

Process	Period	ComputationTime	Priority
a	7	3	3
b	12	3	2
c	20	5	1

$$w_b^0 = 3$$

$$w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$

$$w_b^2 = 6 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

$$R_b = 6$$

24

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

$$R_c = 20$$

Revisit: Process Set C

Process	Period T	ComputationTime C	Priority P	Resp. time R
a	80	40	1	80
b	40	10	2	15
c	20	5	3	5

- The combined utilization is 1.0
- This was above the utilization threshold for three processes (0.78), therefore it failed the test
- The response time analysis shows that the process set will meet all its deadlines
- RTA is necessary and sufficient

CS122A: Embedded System Design, Fall01

Response Time Analysis

- Is sufficient and necessary
- If the process set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a process will miss its deadline
 - Assumption: process takes WCET everytime

CS122A: Embedded System Design, Fall01

Worst-Case Execution Time - WCET

- Obtained by measurement
 - The problem
 - it is difficult to find the worst case
- Obtained by analysis
 - The problem
 - an effective model of the processor (including caches, pipelines, memory wait states and so on) must be available

CS122A: Embedded System Design, Fall01

WCET— Finding C

Most analysis techniques involve two distinct activities.

- The first takes the process and decomposes its code into a directed graph of basic blocks
- These basic blocks represent straight-line code.
- The second component of the analysis takes the machine code corresponding to a basic block and uses the processor model to estimate its worst-case execution time
- Once the times for all the basic blocks are known, the directed graph can be collapsed

CS122A: Embedded System Design, Fall01

Need for Semantic Information

```

for I in 1.. 10 loop
  if Cond then
    -- basic block of cost 100
  else
    -- basic block of cost 10
  end if;
end loop;
    
```

- Simple cost 10*100 (+overhead), say 1005.
- But if Cond only true on 3 occasions then cost is 375

CS122A: Embedded System Design, Fall01

Sporadic Processes

- Sporadic processes have a minimum inter-arrival time
- They also require $D < T$
- The response time algorithm for FPS works here
 - as long as the stopping criteria becomes $W_i^{n+1} > D_i$
- It also works perfectly well with any priority ordering
 - $hp(i)$ always gives the set of higher-priority processes

31

CS122A: Embedded System Design, Fall 01

Hard and Soft Processes

- The worst-case figures for sporadic processes are considerably higher than the averages
 - Interrupts often arrive in bursts
 - an abnormal sensor reading may lead to significant computation
- Measuring schedulability with worst-case figures
 - lead to very low processor utilization figures

32

CS122A: Embedded System Design, Fall 01

General Guidelines

Rule 1 — all processes should be schedulable using average execution times and average arrival rates

Rule 2 — all hard real-time processes should be schedulable using worst-case execution times and worst-case arrival rates of all processes (including soft)

- I
- t may not be possible to meet all current deadlines
 - This condition is known as a transient overload
- Rule 2 ensures that no hard real-time process miss its deadline

33

CS122A: Embedded System Design, Fall 01

Aperiodic Processes

- These do not have minimum inter-arrival times
- Can run aperiodic processes at a priority below the priorities assigned to hard processes
 - they cannot steal, in a pre-emptive system, resources from the hard processes
- For adequate support to soft processes which will often miss their deadlines
 - a server can be employed.
- Servers
 - protect the processing resources needed by hard processes
 - but otherwise allow soft processes to run as soon as possible.

34

CS122A: Embedded System Design, Fall 01

Process Sets with $D < T$

- For $D = T$, Rate Monotonic priority ordering is optimal
- For $D < T$, Deadline Monotonic priority ordering is optimal

$D_i < T_i \quad D_i < T_i$

35

CS122A: Embedded System Design, Fall 01

$D < T$ Example Process Set

Process	Period T	Deadline D	ComputationTime C	Priority P	Response time R
a	20	5	3	4	3
b	15	7	3	3	6
c	10	10	4	2	10
d	20	20	3	1	20

36

CS122A: Embedded System Design, Fall 01

Proof that DMPO is Optimal

- Deadline monotonic priority ordering (DMPO) is optimal if any process set, Q , that is schedulable by priority scheme, W , is also schedulable by DMPO
- The proof of optimality of DMPO involves transforming the priorities of Q (as assigned by W) until the ordering is DMPO
- Each step of the transformation will preserve schedulability

37
CS122A: Embedded System Design, Fall01

DMPO Proof Continued

- Let i and j be two processes (with adjacent priorities) in Q such that under W

$$P_i > P_j \wedge D_i > D_j$$
- Define scheme W' to be identical to W except that processes i and j are swapped

Consider the schedulability of Q under W'

- All processes with priorities greater than P_i will be unaffected by this change to lower-priority processes
- All processes with priorities lower than P_j will be unaffected; they will all experience the same interference from i and j
- Process j , which was schedulable under W , now has a higher priority, suffers less interference, and hence must be schedulable under W'

38
CS122A: Embedded System Design, Fall01

DMPO Proof Continued

- All that is left is the need to show that process i , which has had its priority lowered, is still schedulable
- Under W

$$P_i > P_j \wedge D_i > D_j$$
- Hence process j only interferes once during the execution of i
- It follows that:

$$P_i > P_j \wedge D_i > D_j \implies P_i > P_j \wedge D_i > D_j \wedge P_j > P_i \wedge D_j > D_i$$
- It can be concluded that process i is schedulable after the switch
- Priority scheme W' can now be transformed to W'' by choosing two more processes that are in the wrong order for DMPO and switching them

39
CS122A: Embedded System Design, Fall01

Process Interactions and Blocking

- If a process is suspended waiting for a lower-priority process to complete some computation
 - the priority model is being undermined
- It is said to suffer priority inversion
- If a process is waiting for a lower-priority process,
 - it is said to be blocked
 - This can happen for if there are shared critical resources

40
CS122A: Embedded System Design, Fall01

Priority Inversion

- Consider the executions of four periodic processes
 - a, b, c and d ; and two resources: Q and V

Process	Priority	Execution Sequence	Release Time
a	1	EQQQVE	0
b	2	EE	2
c	3	EVVE	2
d	4	EEQVE	4

41
CS122A: Embedded System Design, Fall01

Example of Priority Inversion

42
CS122A: Embedded System Design, Fall01

Priority Inheritance

- If process p is blocking process q, then q runs with p's priority

CS122A: Embedded System Design, Fall 01

Calculating Blocking

- If a process has m critical sections
 - then the maximum number of times it can be blocked is m
- If B is the maximum blocking time and K is the number of critical sections,
 - the process i has an upper bound on its blocking given by:

$$B_i = \sum_{k=1}^K usage(k, i)C(k)$$

- Under the priority inversion scheme

44

Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$W_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{T_j} \right\rceil C_j$$

45

Priority Ceiling Protocols

Two forms

- Original ceiling priority protocol
- Immediate ceiling priority protocol

46

On a Single Processor

- A high-priority process can be blocked at most once during its execution by lower-priority processes
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured
 - by the protocol itself

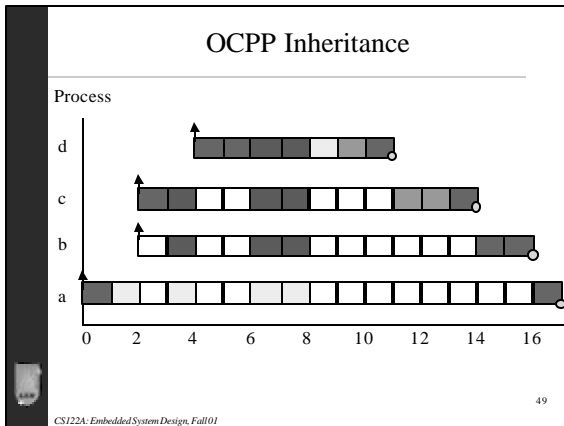
47

OCP

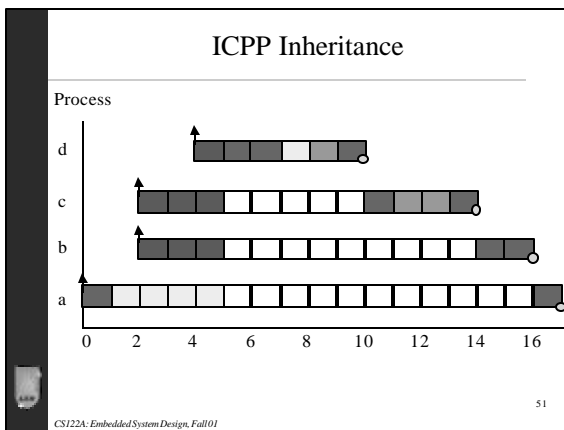
- Each process has a static default priority assigned
 - perhaps by the deadline monotonic scheme
- Each resource has a static ceiling value defined
 - this is the maximum priority of the processes that use it
- A process has a dynamic priority that is
 - the maximum of its own static priority and
 - any it inherits due to it blocking higher-priority processes.
- A process can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource
 - excluding any that it has already locked itself

$$R_i = \max_k usage(k, i)C(k)$$

48



- ### ICPP
- Each process has a static default priority assigned
 - perhaps by the deadline monotonic scheme
 - Each resource has a static ceiling value defined
 - this is the maximum priority of the processes that use it
 - A process has a dynamic priority that is the maximum of
 - its own static priority and
 - the ceiling values of any resources it has locked
 - A process will only suffer a block at the very beginning of its execution
 - Once the process starts actually executing
 - all the resources it needs must be free
 - if they were not, then some process would have an equal or higher priority and the process's execution would be postponed
- 50
- CS122A: Embedded System Design, Fall 01



- ### OCPP versus ICPP
- Although the worst-case behaviour of the two ceiling schemes is identical, there are some points of difference:
 - ICPP is easier to implement than the original (OCPP) as blocking relationships need not be monitored
 - ICPP leads to less context switches as blocking is prior to first execution
 - ICPP requires more priority movements as this happens with all resource usage
 - OCPP changes priority only if an actual block has occurred
- 52
- CS122A: Embedded System Design, Fall 01