
SYSC 3303 Real-Time Concurrent Systems

Scheduling Schemes for Real-Time Systems

- Copyright © 2003 D.L. Bailey and L.S. Marshall, Systems and Computer Engineering, Carleton University
- revised July 30th, 2003

Correctness of Concurrent Programs

- In a non-real-time concurrent system, as long as the algorithms in each process are implemented correctly, and as long as the required mutual exclusion and condition synchronization are correctly programmed, the outputs of the program will be the same regardless of the order in which the processes are executed
 - in these slides, the terms process and thread are synonymous
- As long as the program's outputs are correct, the timing behaviour is not a major issue
- Real-time systems are different

What is a Real-Time System?

- “A real-time system ***must respond to unpredictable stimuli from its environment in a timely fashion*** while operating reliably and continuously in the presence of failures in its own (perhaps distributed) components and connections, and uncertainty about the state of its environment.”

Hard vs. Soft Real-Time

- *Hard* real-time system - responses must occur within specified deadlines
 - the system fails if its deadlines are missed, and this failure can have catastrophic results
- *Soft* real-time system - response times are important, but system functions correctly if a deadline is occasionally missed (e.g., responses occasionally delivered late or not at all)
- In comparison, a system is called *interactive* if it does not have specified deadlines but attempts to provide "adequate" response times

Process Scheduling in R-T Systems

- In other words, real-time systems have *temporal requirements* in addition to their other requirements
- Processes in a real-time system must be scheduled so that the system meets its temporal requirements
- A scheduling scheme provides
 - an algorithm for ordering the use of system resources (especially the CPU(s))
 - a means of predicting the worst-case behaviour of the system when the scheduling algorithm is applied

What We've Seen So Far

- Earlier in the course, we discussed prioritized, preemptive scheduling, and mechanisms for controlling priority inversion (priority inheritance protocol, priority ceiling protocol), but our descriptions were qualitative in nature
- Now we'll go back to first principles, and provide a quantitative description of various scheduling schemes
- To simplify the analysis of the worst-case system behaviour, we'll initially use a very simple process model when presenting some standard scheduling schemes

A Simple Process Model

- The program consists of a fixed set of processes
- All processes are periodic, with known periods
- All processes are completely independent of each other
 - as such, at some point in time all processes will become ready-to-run at the same instant
- All system overheads (e.g., context switching times) are assumed to be 0
- All processes have a deadline equal to their period
- All processes have a fixed worst-case execution time

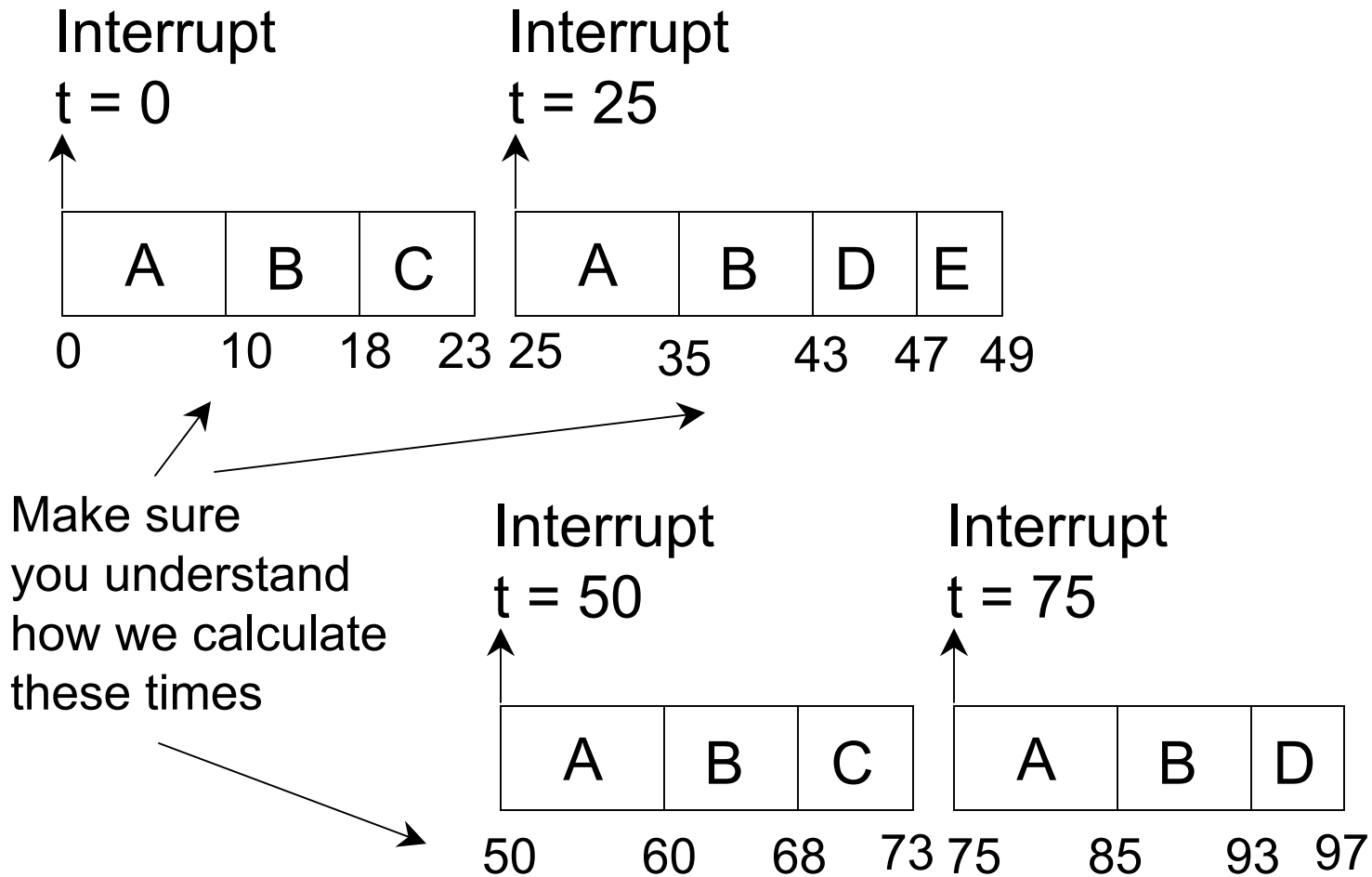
Cyclic Executive

- We plan a complete schedule such that the repeated execution of this schedule causes all processes to run at their correct rate
- Example:

Process	Period, T	Computation time, C
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2

- Here is one possible schedule:

Cyclic Executive



Cyclic Executive

```
for (;;) {  
    wait_for_interrupt;  
    A;  
    B;  
    C;  
    wait_for_interrupt;  
    A;  
    B;  
    D;  
    E;  
}
```

} minor cycle

} minor cycle

Cyclic Executive

```
wait_for_interrupt;  
A;  
B;  
C;  
wait_for_interrupt;  
A;  
B;  
D;  
}
```

} minor cycle

} minor cycle

Cyclic Executive

- The complete loop body is known as a *major cycle*
- A major cycle is made up of *minor cycles*
- The occurrence of a timer interrupt triggers the execution of the next minor cycle
- Here, there are 4 minor cycles of 25 ms duration (the time between interrupts), making up a 100 ms major cycle
- We should verify that, with this schedule, the processes are executed at the correct rate

Cyclic Executive

- Process A, $T = 25$ ms
 - time between 2nd and 1st executions = $25 - 0 = 25$
 - time between 3rd and 2nd executions = $50 - 25 = 25$
 - time between 4th and 3rd executions = $75 - 50 = 25$
 - time between 5th and 4th executions = 100 (start of 2nd major cycle, not shown) - $75 = 25$
- Process B, $T = 25$ ms
 - time between 2nd and 1st executions = $35 - 10 = 25$
 - time between 3rd and 2nd executions = $60 - 35 = 25$
 - time between 4th and 3rd executions = $85 - 60 = 25$
 - time between 5th and 4th executions = 110 (in 2nd major cycle) - $85 = 25$

Cyclic Executive

- Process C, $T = 50$ ms
 - time between 2nd and 1st executions = $68 - 18 = 50$
 - time between 3rd and 2nd executions = 118 (in 2nd major cycle, not shown) - 68 = 50
- Process D, $T = 50$ ms
 - time between 2nd and 1st executions = $93 - 43 = 50$
 - time between 3rd and 2nd executions = 143 (in 2nd major cycle, not shown) - 93 = 50
- Process E, $T = 100$ ms
 - time between 2nd and 1st executions = 147 (in 2nd major cycle, not shown) - 47 = 100

Cyclic Executive - Important Features

- Notice that each process can be implemented by a function/procedure, and that there is none of the context-switching overhead normally associated with concurrent processes - the scheduler executes each "process" by performing a function call, and each minor cycle is just a sequence of function calls
- Functions share a common address space, and can pass data between themselves (no need to program mutex because concurrent access not possible)
- All "process" periods must be a multiple of the minor cycle time

Cyclic Executive - Drawbacks

- It is difficult to include aperiodic (sporadic) processes
- Extra work is required to incorporate processes with long periods
 - the major cycle time is normally the maximum period that can be accommodated
 - to accommodate processes with periods longer than the major cycle, we can add a procedure to the major cycle that will call a secondary procedure every N major cycles

Cyclic Executive - Drawbacks

- Each cyclic executive must be built "from scratch"
- A "process" with a long computation time (compared to the other processes) may need to be split into a fixed number of fixed sized procedures, which are called from different minor cycles
 - this may cut across the structure of the code from a software engineering perspective, so may be error-prone or difficult to maintain

Cyclic Executive

- If it is possible to construct a cyclic executive for a set of periodic processes, then no further schedulability test is required (the scheme is "proof by construction")
- For systems with a large number of processes, building the executive is problematic
- An analogy can be made with the classic bin-packing problem: items of varying sizes (in one dimension) have to be placed in the minimum number of bins such that no bin is over-full
- The bin-packing problem is NP-hard, so is computationally infeasible for sizeable problems

Process-Based Scheduling

- A more flexible approach than a cyclic executive is to provide an environment (e.g., a real-time kernel or real-time o/s) that supports process creation and execution
- Processes can be in one of a number of states; e.g.,
 - runnable
 - suspended waiting for a timing event
 - suspended waiting for a non-timing event (appropriate for sporadic processes)

Process-Based Scheduling

- Each process has a priority attribute
 - in a real-time system, a process' priority is determined by its temporal requirements, not its importance
- Runnable processes are executed in the order determined by their priority
- A high-priority process may become runnable during the execution of a low-priority process
- In a preemptive, priority-based scheduling scheme, an immediate context switch to the higher-priority process occurs

Process-Based Scheduling

- In a **non-preemptive** scheduling scheme, the lower priority process is allowed to run until it relinquishes the processor, then the higher priority process executes
- Between these extremes, we can have **deferred preemption**
 - after the higher-priority process becomes runnable, the lower-priority process is allowed to continue executing for a bounded time
 - if it has not relinquished the processor by the end of that time, it is preempted

Rate Monotonic Priority Assignment

- Rate monotonic priority assignment is a priority assignment scheme for the simple process model described earlier
- Each process is assigned a unique priority based on its period
 - the shorter the period T , the higher the priority P
 - if $T_i < T_j$, $P_i > P_j$

Rate Monotonic Priority Assignment

- **Example:**

Process	Period, T	Priority, P
F	25	5
G	60	3
H	42	4
I	105	1
J	75	2

- Note that priority 1 is the lowest, priority 5 is the highest

Utilization-Based Schedulability Tests

- In 1973, Liu and Layland presented a simple test for schedulability when rate monotonic priority assignment is used
- This test considers only the utilization of the process set
 - utilization, U , of a process equals its computation time divided by its period: C / T
- In a system with N processes, all processes will meet their deadlines if the following condition holds:

$$\sum_{i=1}^N \left(\frac{C_i}{T_i} \right) < N \left(2^{1/N} - 1 \right)$$

Utilization-Based Schedulability Tests

N	Utilization Bound (%)
1	100
2	82.8
3	78.0
4	75.7
5	74.3
10	71.8

- The bound asymptotically approaches 69.3%
- Any process set with a total utilization of less than 69.3% will always be schedulable by a preemptive priority-based scheduling scheme, with priorities assigned by the rate monotonic algorithm

Utilization-Based Schedulability Tests

- **Example 1**

Process	T	C	P	U
P1	50	12		
P2	40	10		
P3	30	10		

- **Assign priorities (RMA) and calculate utilization:**

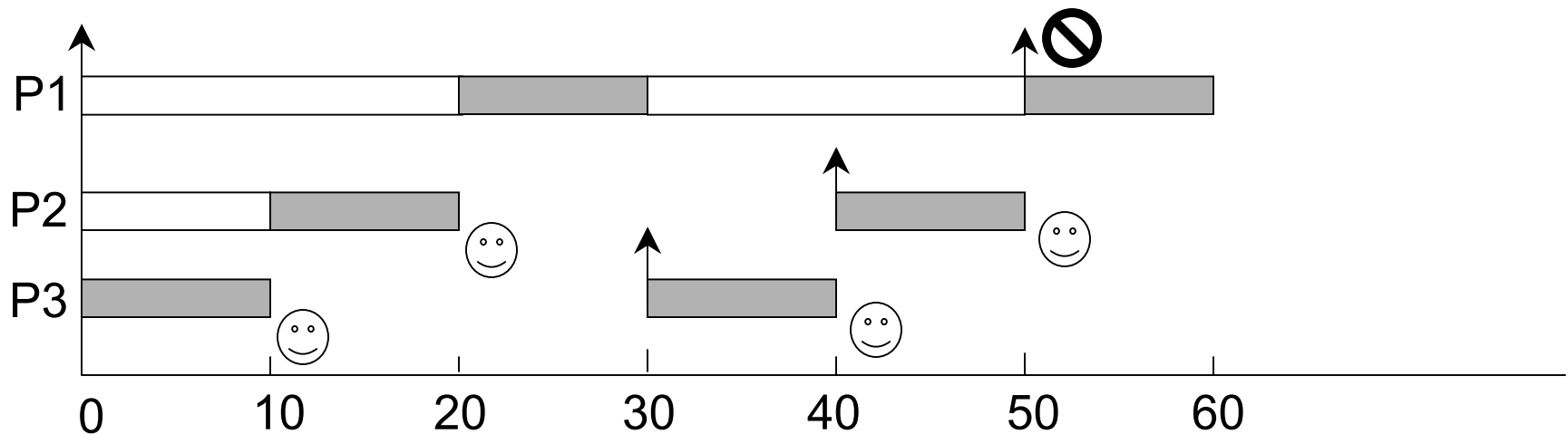
Process	T	C	P	U
P1	50	12	1	0.24
P2	40	10	2	0.25
P3	30	10	3	0.33

- **Combined utilization is 0.82 (82%), which is above the bound for three processes (78%)**

Utilization-Based Schedulability Tests

- Therefore, this process set fails the utilization test
- Does this mean that the processes are not schedulable by a preemptive, priority-based scheme so that they meet their deadlines?
- No - we need to do more analysis
- We can use a timeline to depict the behaviour of this process set (assume all processes runnable at $t = 0$)
- Notice, on the next slide, that P1 has executed for only 10 units before its deadline at $t = 50$
 - it needed 12 units of computation time, so it missed its first deadline

Utilization-Based Schedulability Tests



Start of period



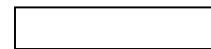
Process completion time
- deadline met



Deadline missed



Executing



Preempted

Verifying Schedulability with Timelines

- Timelines can be used on their own to test for schedulability
- How long a timeline must be drawn to verify that the process set will always meet its deadlines?
- Liu and Layland showed that for processes that are all runnable at $t = 0$, it can be shown that a timeline equal to the length of the longest period is sufficient
 - if all processes meet their first deadlines they will meet all future ones

Utilization-Based Schedulability Tests

- **Example 2**

Process	T	C	P	U
P4	80	32		
P5	40	5		
P6	16	4		

- **Assign priorities (RMA) and calculate utilization:**

Process	T	C	P	U
P4	80	32	1	0.400
P5	40	5	2	0.125
P6	16	4	3	0.250

- **Combined utilization is 0.775 (77.5%), which is below the bound for three processes (78%)**

Utilization-Based Schedulability Tests

- Therefore, this process set passes the utilization test, and the processes are guaranteed to meet all their deadlines
- Exercise for the student: draw the timeline for this process set

Utilization-Based Schedulability Tests

- **Example 3**

Process	T	C	P	U
P7	80	40		
P8	40	10		
P9	20	5		

- **Assign priorities (RMA) and calculate utilization:**

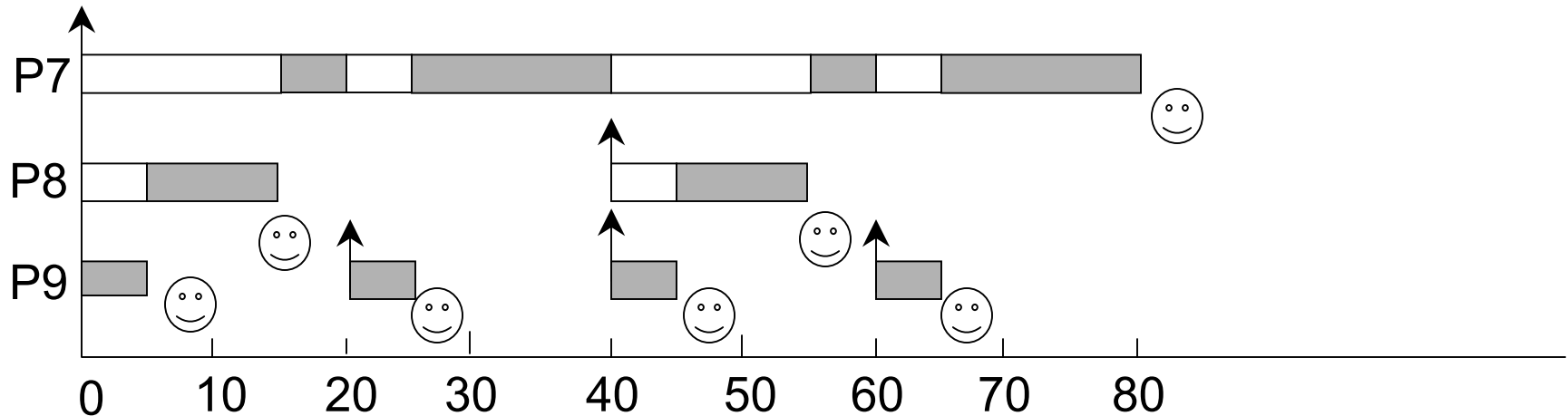
Process	T	C	P	U
P7	80	40	1	0.50
P8	40	10	2	0.25
P9	20	5	3	0.25

- **Combined utilization is 1.00 (100.0%), which is above the bound for three processes (78%)**

Utilization-Based Schedulability Tests

- Therefore, this process set fails the utilization test
- But, as the timeline on the following slide shows, all deadlines are met up to $t = 80$ (the longest period) so the processes are guaranteed to meet all their deadlines
- The utilization test is **sufficient** but not **necessary**
- If a process set passes the test, it will meet all its deadlines
- If a process set fails the test, it may or may not fail at run-time

Utilization-Based Schedulability Tests



Start of period



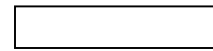
Process completion time
- deadline met



Deadline missed



Executing



Preempted