Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment Summary

Chengyu Liu, Jaeeun Kim, Anya Svintsitski and Tiancheng Zhang

Intro

- Combinatoric scheduling analyze for pure process control
- No non-time-critical jobs exist
- Need to carefully schedule time-critical control and monitor functions to achieve the efficient use of computer

Examples of time-critical tasks

- Pointing of an antenna to track a spaceship orbit
- Autopilot
- Multiprocessor
- Flight control software

Multiprocessor

- Transistor leakage current becomes important under sub-100 nm technology
- Reducing voltage limits the maximum operating frequency
- Multiprocessor real-time scheduling is a much more difficult problem than uniprocessor scheduling

Multiprocessor

- Heterogeneous: the processors are different
- Homogeneous: the processors are identical
- Uniform: the rate of execution of a task depends only on the speed of the processor

Scheduling algorithm

- Fixed priority assignment (utilization is 70%)
- Dynamic assignment of priorities (can achieve full utilization)
- Both are priority driven and preemptive (the processing of any task is interrupted by a request for any higher priority task)

Background

- Tasks are executed in response to events in the equipment controlled or monitored by the computer.
- The remainder are executed in response to events in other tasks.
- Each of the tasks must be completed before some fixed time has elapsed following the request for it.

Prior Work

- Manacher derives an algorithm for the generation of task schedules in a hard-real-time environment, but it is restricted to the somewhat unrealistic situation of only one request time for all tasks.

- Lampson proposes a program based on the timing information supplied for programs needing guaranteed service.

- Martin depicts the range of systems which is "real-time"

The Environment

5 Assumptions of Program Behavior in hard-real-time

- 1 : The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.
  - Valid for pure process control
- 2 : Deadlines consist of run-ability constraints only--i.e, each task must be completed before the next request for it occurs.
  - Able to eliminates queuing problems but significant amount of buffering hardware must exist to hold
- 3 : The tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks.
- 4: Run-time for each task is constant for that task and does not vary with time. Run-time here refers to the time which is taken by a processor to execute the task without interruption.
  - Maximize processing time for a task. Benefit from existence of large main memories
- 5: Any nonperiodic tasks in the system are special; they are initialization or failure-recovery routines; they displace periodic tasks while they themselves are being run, and do not themselves have hard, critical deadlines.

Scheduling Algorithm

- Rules that determine the task to be executed at a particular moment with pre-emptive and priority driven
- Method of assigning priorities to task

  Fixed priority scheduling algorithm
  - Static scheduling algorithm due to assigned priorities

  Mixed scheduling algorithm
  - Dynamic scheduling algorithm because priorities of task can be changed from request to request

Fixed Priority Scheduling Algorithm

Fixed Priority Scheduling Algorithm

THEOREM 1. A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks

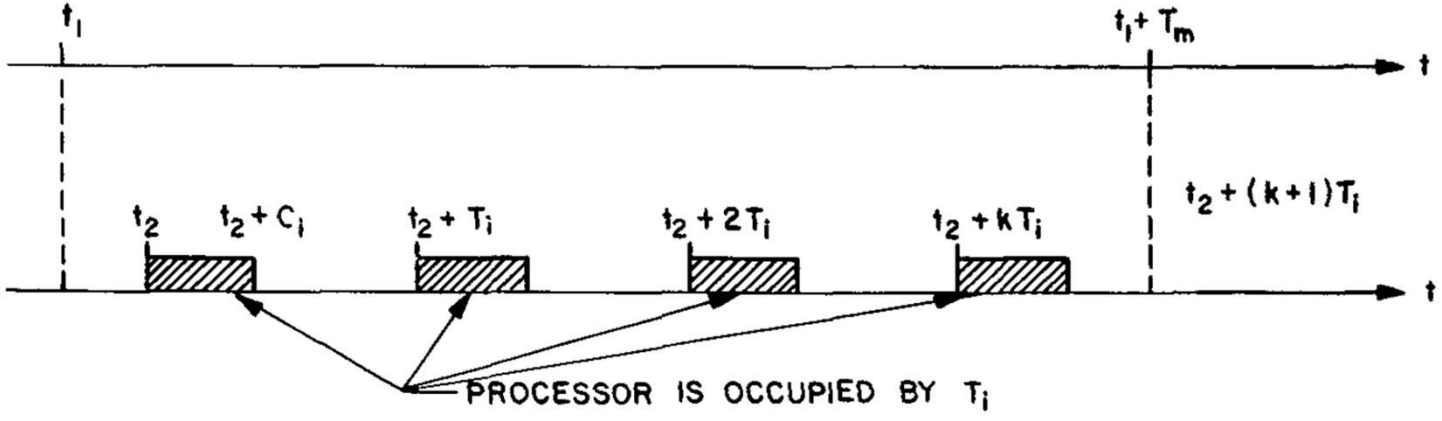


FIG. 1. Execution of τ_i between requests for τ_m

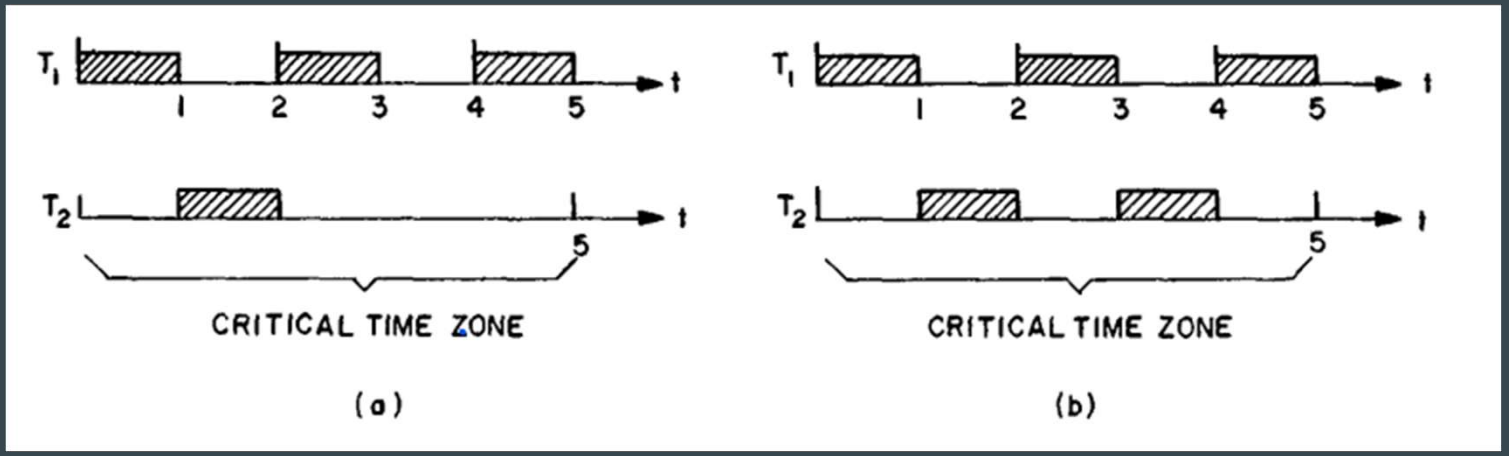
τ: Task
T: Request Period
C: Run time

τ_m: lowest priority task
τ_i: higher priority task than τ_m
t1: τ_m occurs at t1
t2: τ_i occurs at t2

- Pre-emption of τ_m by τ_i causes delay in the completion of the request for τ_m
- t2 will not speed up the completion of τ_m(unchanged or delayed)
- When t2 coincide with t1, τ_m is the largest

Fixed Priority Scheduling Algorithm

Examples:

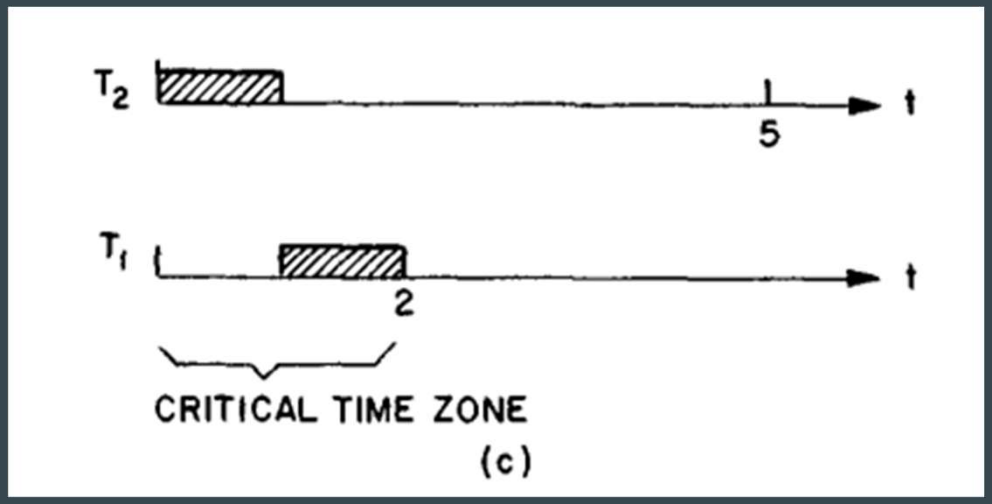


τ_1: higher priority task than τ_2
τ_2: lower priority task than τ_1
T1 = 2
T2 = 5
C1 = 1
C2 = 1

- Priority assignment is feasible
- C2 can be increased at most to 2

Fixed Priority Scheduling Algorithm

Examples:



τ_1: higher priority task than τ_2
τ_2: lower priority task than τ_1
T1 = 2
T2 = 5
C1 = 1
C2 = 1

- In case of τ_2 is the higher priority task, C1 and C2 cannot exceed 1
- Feasible with τ_1 at higher priority than τ_2, but the opposite is not true

Fixed Priority Scheduling Algorithm

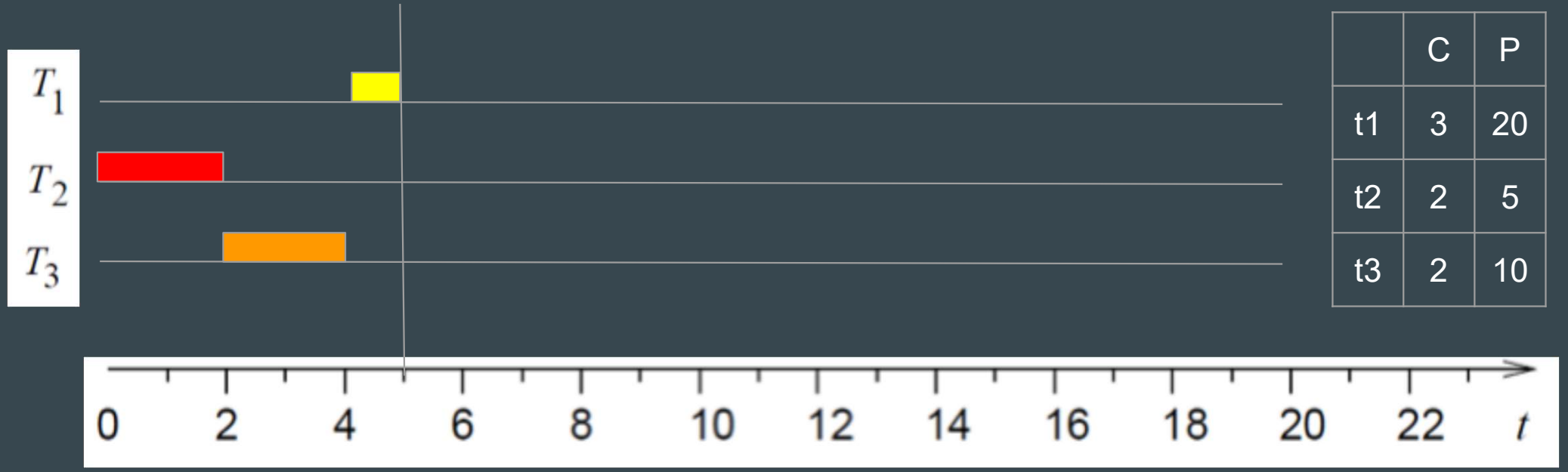
THEOREM 2. If a feasible priority assignment exists for some task set, the rate monotonic priority assignment is feasible for that task set

- Rate-monotonic priority assignment: tasks with higher request rates will have higher priorities
- Priority assignment is optimum when no other fixed priority assignment cannot be scheduled by rate-monotonic priority assignment

Rate Monotonic Scheduling example

- Request rate is reciprocal of request period
- Higher request rates has higher priorities

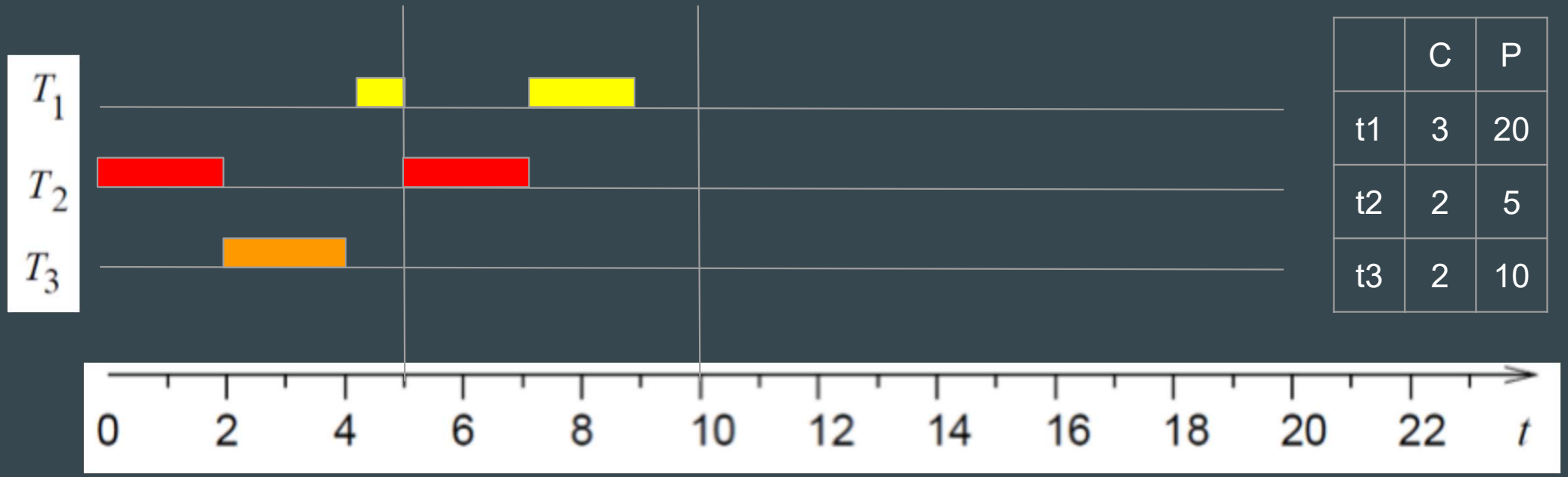
Based on the request rates, priority is set to t2>t3>t1



	C	P
t1	3	20
t2	2	5
t3	2	10

Rate Monotonic Scheduling example

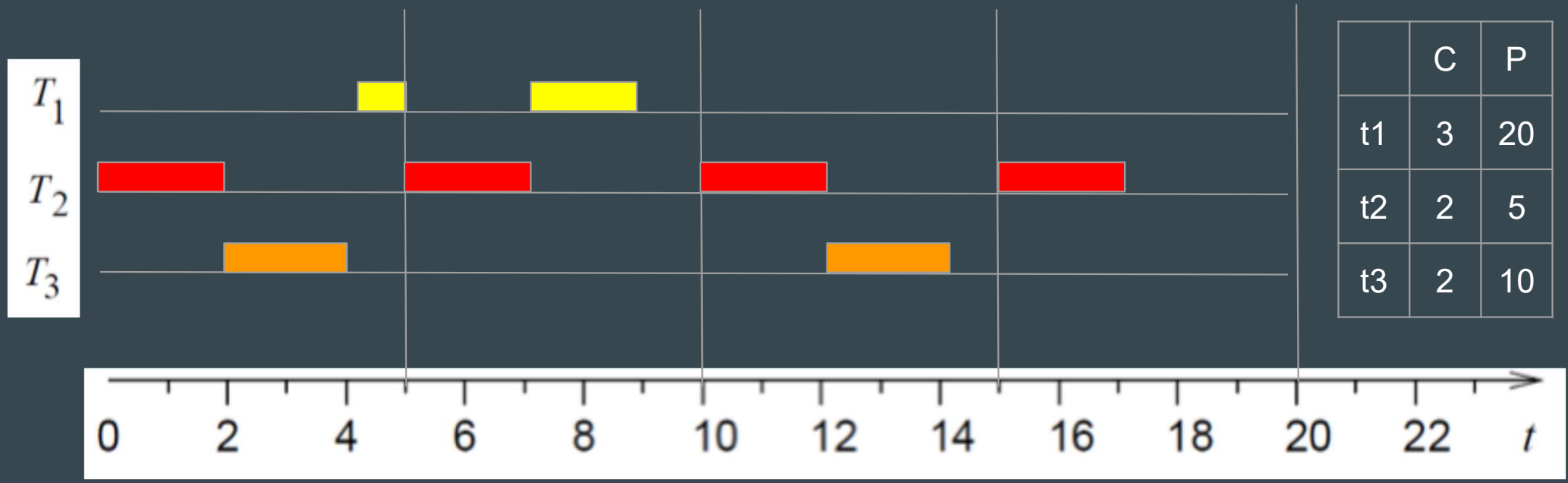
priority is set to t2>t3>t1



	C	P
t1	3	20
t2	2	5
t3	2	10

Rate Monotonic Scheduling example

priority is set to t2>t3>t1



	C	P
t1	3	20
t2	2	5
t3	2	10

Resultant priority assignment is still feasible

Achievable Processor Utilization

Achievable Processor Utilization

Utilization Factor

- Processor time execution on the task set.
  - Fully Utilize
  - Least Upper bound
- Rate-monotonic Priority Assignment

$$U = \sum_{i=1}^{m} (C_i/T_i).$$

Achievable Processor Utilization

THEOREM 3. For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is U = 2*(2^(1/2)-1)

THEOREM 4. For a set of m tasks with fixed priority order, and the restriction that the ratio between any two request periods is less than 2, the least upper bound to the processor utilization is U = m(2^(1/m)-1).

THEOREM 5. For a set of m tasks with fixed priority order, the least upper bound to processor utilization is U = m (2^(1/m) - 1).

Achievable Processor Utilization

THEOREM 3. For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is U = 2*(2^(1/2)-1)

Assumption: T2>T1, and τ1 has higher priority than τ2

Case 1: C1 is short enough that all requests for critical zone of T2 are completed before the second τ2 request.

The corresponding utilization factor is:

$$U = 1 + C_1[(1/T_1) - (1/T_2) \lceil T_2/T_1 \rceil].$$

Achievable Processor Utilization

THEOREM 3. For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is U = 2*(2^(1/2)-1)

Assumption: T2>T1, and τ1 has higher priority than τ2

Case 2: The execution of the T2/T1 th request for t1 overlaps the request for τ2

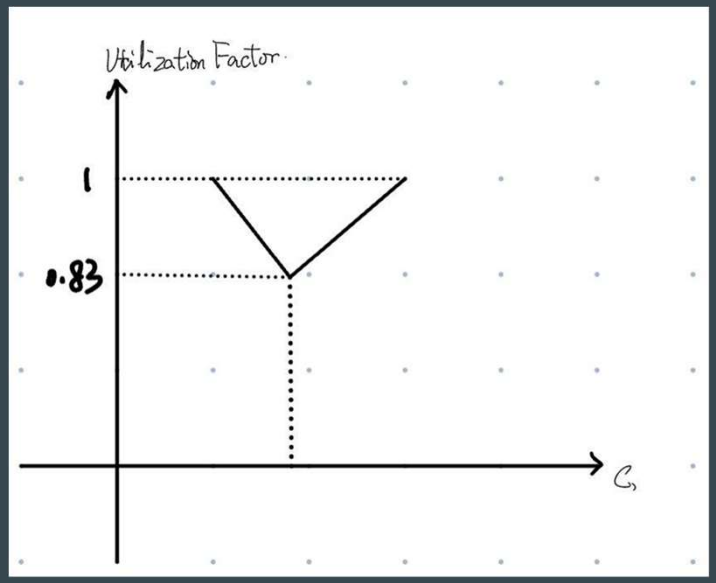
The corresponding utilization factor is:

$$U = (T_1/T_2) \lfloor T_2/T_1 \rfloor + C_1[(1/T_1) - (1/T_2) \lfloor T_2/T_1 \rfloor].$$

Achievable Processor Utilization

THEOREM 3. For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is U = 2*(2^(1/2)-1)

Plot of the utilization factor with respect to C1



Achievable Processor Utilization

THEOREM 3. For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is U = 2*(2^(1/2)-1)

The boundary is achieved when

$$C_1 = T_2 - T_1 \lfloor T_2/T_1 \rfloor$$

The least bound U=2*(2^(1/2)-1) is achieved when {T2/T1} =2^(1/2)-1;

When {T2/T1}=0, the U can goes to 1;

Achievable Processor Utilization

THEOREM 4. For a set of m tasks with fixed priority order, and the restriction that the ratio between any two request periods is less than 2, the least upper bound to the processor utilization is $U = m(2^{(1/m)}-1)$.

Step 1: A proof that when $Ci = T(i+1)-T(i)$ will achieve the least upper bound for the processor utilization.

Method: For any $Ci \neq T(i+1)-Ti$,

prove that a pair of $(Ci', C(i+1)') = (T(i+1)-Ti, C(i+1)+\Delta)$,

can make the U smaller.

Achievable Processor Utilization

THEOREM 4. For a set of m tasks with fixed priority order, and the restriction that the ratio between any two request periods is less than 2, the least upper bound to the processor utilization is $U = m(2^{(1/m)}-1)$.

Step 2: Given $C_i = T_{(i+1)}-T_{(i)}$, use the partial differential equation to achieve the minimum value of U:

Method: let $g_i = (T_m - T_i)/T_i$, $i = 1, 2, \ldots, m$.

The least upper bound of utilization factor $U = m(2^{(1/m)}-1)$ achieved, When $g_i = 2^{((m-i)/m)}-1$

Achievable Processor Utilization

THEOREM 5. For a set of m tasks with fixed priority order, the least upper bound to processor utilization is $U = m (2^{(1/m)} - 1)$.

Implication:
For $m = 3$, $U = 0.78$
For large m, $U = \ln(2) = 0.693$

Relaxing the Utilization Bound

Relaxing the Utilization Boundary

Methods to improve the utilization:

- Make {Ti/Tj}=0, refer to the Theorem 3
- Buffer several low-priority tasks and relax their hardlines.
  - Reasonable execution fashion (e.g. FIFO)
  - Finite task period
- Dynamic task priority assignment

The Deadline Driven Scheduling Algorithm

- Also known as Earliest deadline first (EDF)
- Task lengths/task periods ≤ 1, else overflow due to no available processor time
  - **THEOREM 6:** When the deadline driven scheduling algorithm is used to schedule a set of tasks on a processor, there is no processor idle time prior to an overflow.
  - **THEOREM 7**: For a given set of m tasks, the deadline driven scheduling algorithm is feasible if and only if $(C_1/T_1) + (C_2/T_2) + ... + (C_m/T_m) \leq 1$.

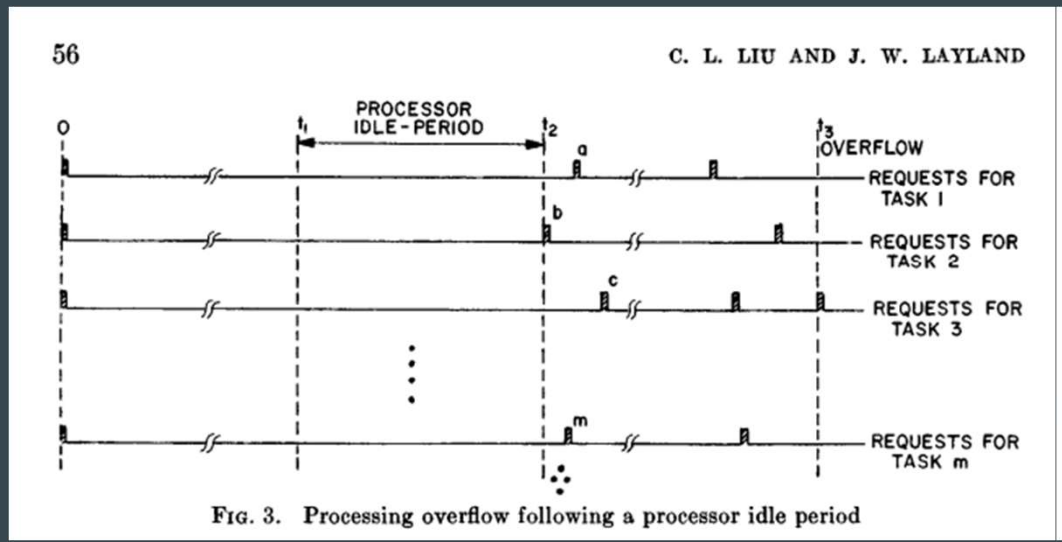
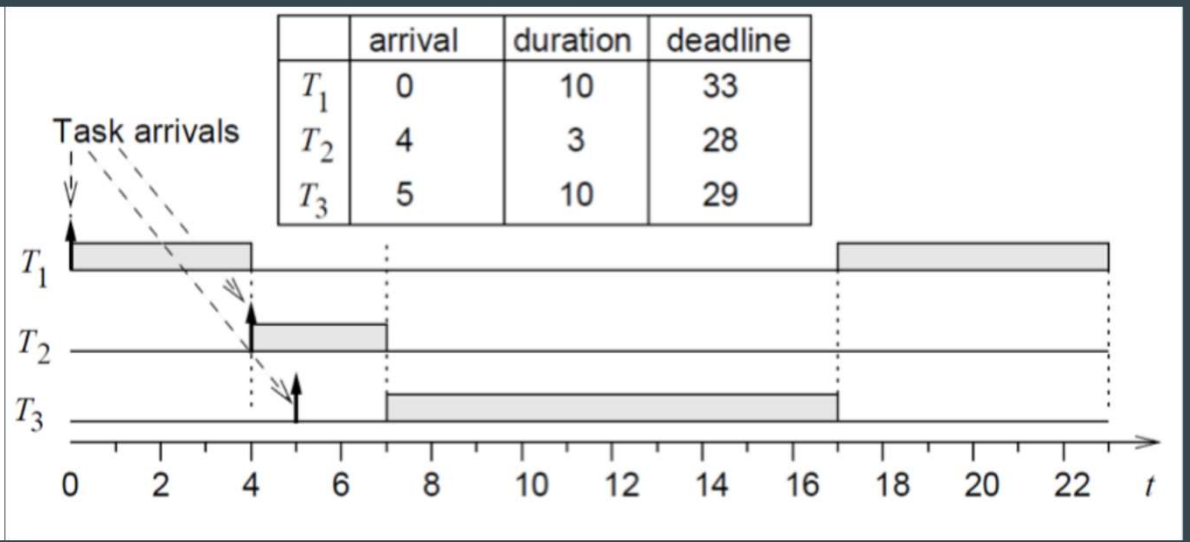
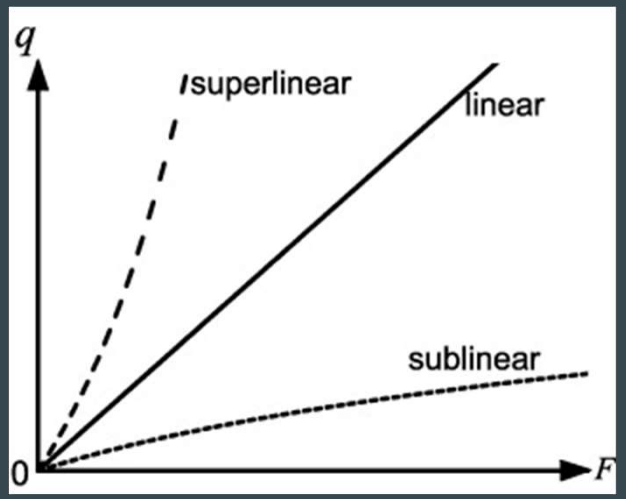


FIG. 3. Processing overflow following a processor idle period



A Mixed Scheduling Algorithm

- Tasks 1-arbitrary k get RM scheduled, k+1 - m get EDF scheduled
- For nondecreasing a(t), $a(T) \leq a(t + T) - a(t)$ means a(t) is sublinear
- $a_k(t)$ is the availability function of the processor for tasks k+1, k+2...m
  - Can't decrease, availability can't be removed
  - Sublinear by critical timezone argument
- **THEOREM 8:** If a set of tasks are scheduled by the deadline driven scheduling algorithm on a processor whose availability function is sublinear, then there is no processor idle period to an overflow



Compare

- Mixed scheduling algorithm

$$U = \frac{1}{3} + \frac{1}{4} + \frac{2}{5} = 98.3\%.$$

- Fixed priority scheduling algorithm

$$U = \frac{1}{3} + \frac{1}{4} + \frac{1}{5} = 78.3\%,$$

Compare

- This example strongly suggests that the bound is considerably less restrictive for the mixed scheduling

- Algorithm than for the fixed priority rate-monotonic scheduling algorithm. The mixed scheduling algorithm may thus be appropriate for many applications.

Conclusion

- The most important and least defensible of these are (A1), that all tasks have periodic requests, and (A4), that run-times are constant.

- A combination of the two scheduling algorithms appears to provide most of the benefits of the deadline driven scheduling algorithm.

Thank You!

Any questions?