SYSC 5701
Operating System Methods for Real-Time Applications

Scheduling Aperiodic and Sporadic Jobs in Priority Driven Systems

Winter 2014

Liu Ch. 7

Assumptions

1. now allow aperiodic or sporadic tasks !!
2. tasks are independent (still)
   - will relax assumption later
     - interdependency → Liu Ch. 8
3. uniprocessor

Aperiodic Events

- **aperiod·ic**: *a.* having no natural frequency
- stochastic inter-arrival times
- some may be critical – "sporadic" period
- **sporad·ic**: *a.* intermittent; scattered; single
- will consider techniques:
  - background
  - polling
  - sporadic server

Aperiodic Service in Background

- service aperiodic tasks as “background” process
- lowest priority – **always preempt** for periodic tasks
- OK for non-critical, less important activities
- may cause aperiodic deadlines to be missed
- could starve aperiodic work

Background (con't)

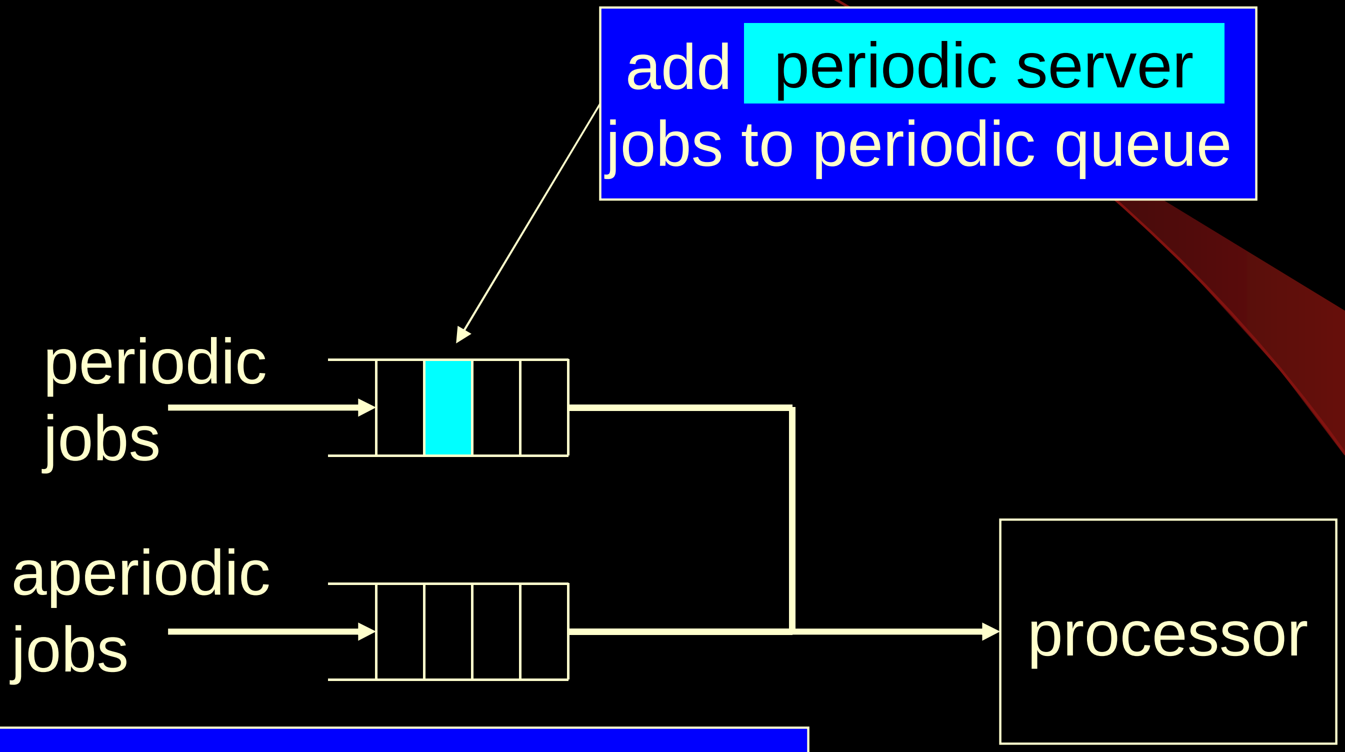
- may cause poor response time for aperiodic work
- response time =

  time of completed service – time of request
- <u>best case</u>: no pending periodic work
- <u>worst case</u>: all periodic work pending
- background → OK for some cases, but …

Integrate with Periodic?

- how to ensure aperiodic work not starved with RM approach?
- must cast aperiodic tasks into periodic framework
- must know task inter-arrival characteristics
- allocate a periodic task that does nothing but service aperiodic work!

Priority Queues in Kernel



periodic server jobs
process aperiodic jobs

Aperiodic Service Using Polling

- **goal**: improve response time to aperiodic requests
  - schedule periodic "aperiodic service" as a regular server task: period = p_s, budget = e_s
- when aperiodic job released
  - → put it in aperiodic job queue
- when aperiodic server executes:
  - polls aperiodic job queue and execute requests
  - won't exceed allocated budget

Polling (con't)

- if all pending aperiodic jobs completed before consuming budget:
  - server exhausts remaining budget: set = 0
  - suspends itself
- budget replenished at beginning of each period

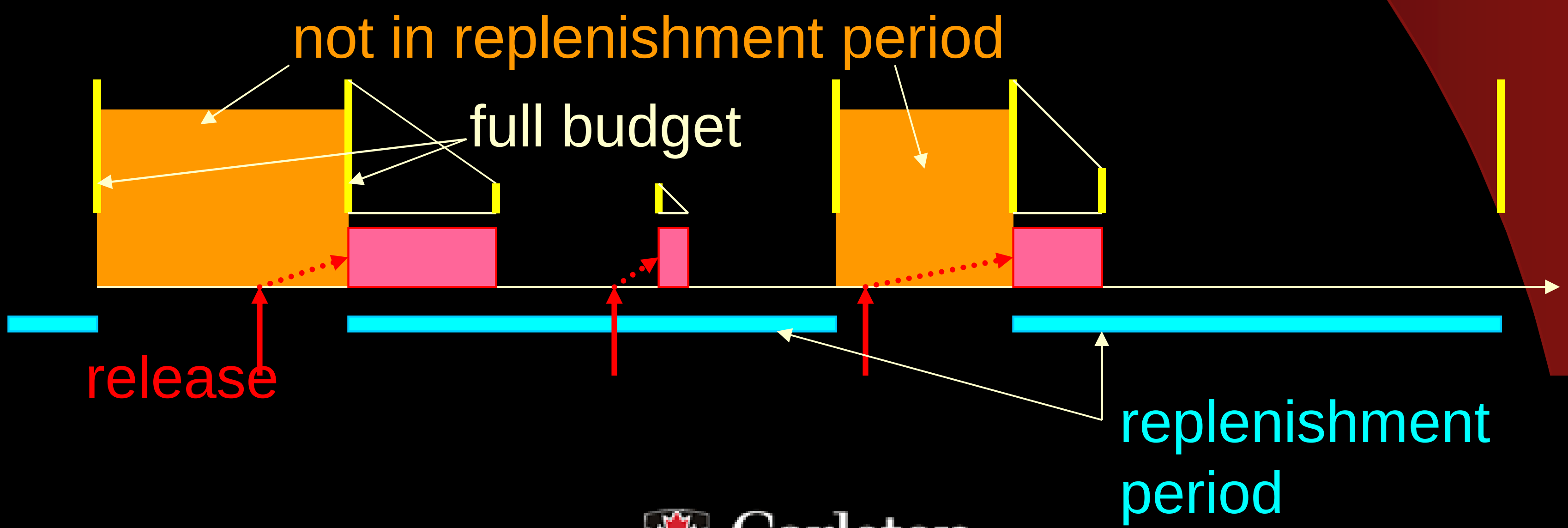
- What if job arrives just after server suspends self, even though budget existed for job?
- What to do if aperiodic job misses deadline?
- How to schedule aperiodic jobs in aperiodic queue?

Bandwidth-Preserving "Deferrable" Server

- server does not exhaust budget before suspending self
- budget replenished to e_s at start of each period
- allows "late" arrivals to execute without waiting until next period
- improves response time of aperiodic jobs

“Deferred” (vs. Deferrable) Server

- different replenishment algorithm!
- has “sliding window” replenishment period
- replenishment period starts when server begins execution with a full budget



Performance ?

[Sprunt, Lehoczky, Sha]

- deferred server response time improvement:
  - **6** times better than polling
  - **10** times better than background
- o/s implementation issues:
  - more complex
  - hybrid: static + dynamic scheduling
  - managing server execution capacity

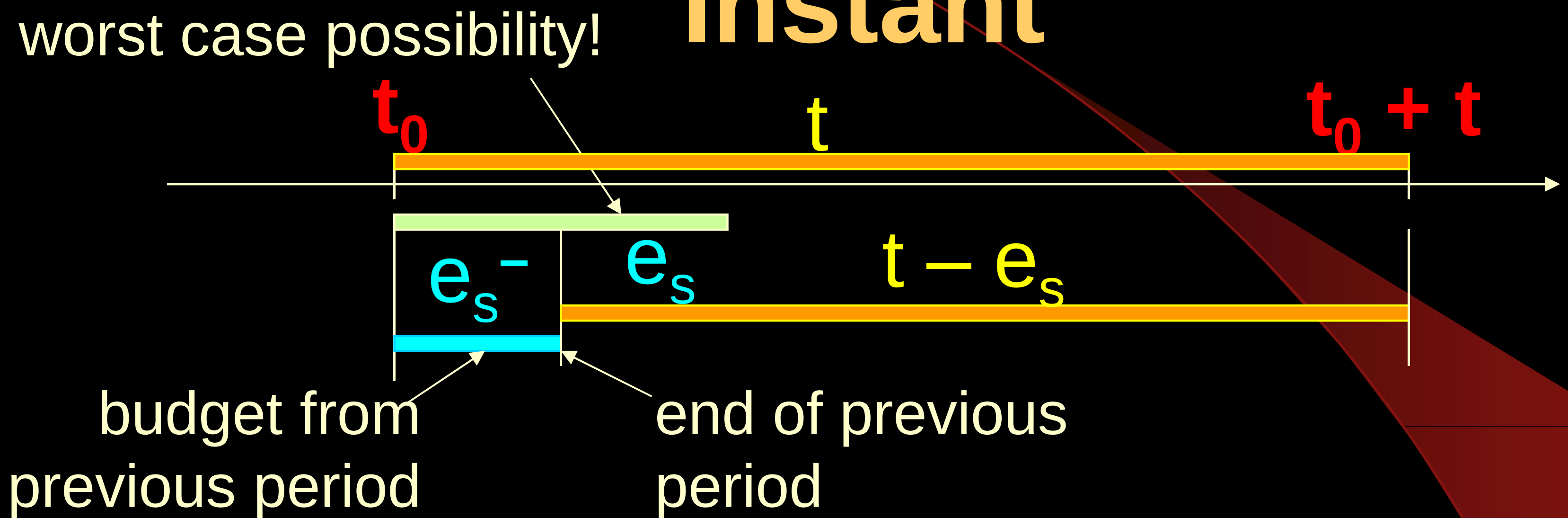
Theory?

for deferred server delaying lower priority jobs:

- worst case critical instant:
  1. server is highest priority ready job
  2. has almost full budget (e_s^-)
  3. e_s^- remaining in period

     → will be replenished after e_s^-

Deferred Server Critical Instant



- over time t from critical instant t_0, server can delay lower priority tasks by:

$$e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s \Rightarrow \left(1 + \left\lceil \frac{t - e_s}{p_s} \right\rceil \right) e_s$$

Equation 7.1 in Liu Text

$$\omega_i(t) = e_i + e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s + \sum_{k=0}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k$$

max. delay due to server

delay due to higher priority periodic jobs

Schedulable Utilization

- Theorem 7.2 in text:

for n periodic tasks

$$U_{RM/DS}(n) = (n-1)\left[\left(\frac{u_s+2}{u_s+1}\right)^{\frac{1}{(n-1)}} - 1\right]$$

where $u_s = \frac{e_s}{p_s}$

Carleton UNIVERSITY

Server Priority

- if server priority < priority of task i
  → server does not influence response time of task i
- if server priority > priority of task i
  → use Equation 7.1

Multiple Servers?

- might prefer to have different priorities associated with different aperiodic tasks
- use a server for each priority level
- if m servers with priority > priority of task i:

$$\omega_i(t) = e_i + \sum_{k=1}^{m}\left(1 + \left\lceil \frac{t - e_{s,k}}{p_{s,k}} \right\rceil\right) e_{s,k} + \sum_{k=0}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k$$

Carleton UNIVERSITY

Sporadic Server

- Sporadic → hard deadlines (vs. aperiodic)
- worst case → replenish server at the end of every sporadic server period
  - essentially the same as deferrable server
- acceptance test?
  - only accept a job if enough slack budget to complete job before deadline

Fixed Priority Acceptance Test

- assume sporadic jobs placed in sporadic job queue in EDF order
- slack σ of a <u>job</u> = server budget left over after executing the job
  - Can be influenced by other jobs accepted by server!

Carleton UNIVERSITY

Slack of a Job

- for job $S_i(t , d_{s,i} , e_{s,i})$:

$$\sigma_{s,i}(t) = \left\lfloor \frac{d_{s,i} - t}{p_s} \right\rfloor e_s - e_{s,i} - \sum_{d_{s,k} < d_{s,i}} (e_{s,k} - \xi_{s,k})$$

execution of previously accepted jobs with earlier deadlines

number of replenishments before deadline → conservative !

server period

server budget

completed portion of job

Acceptance

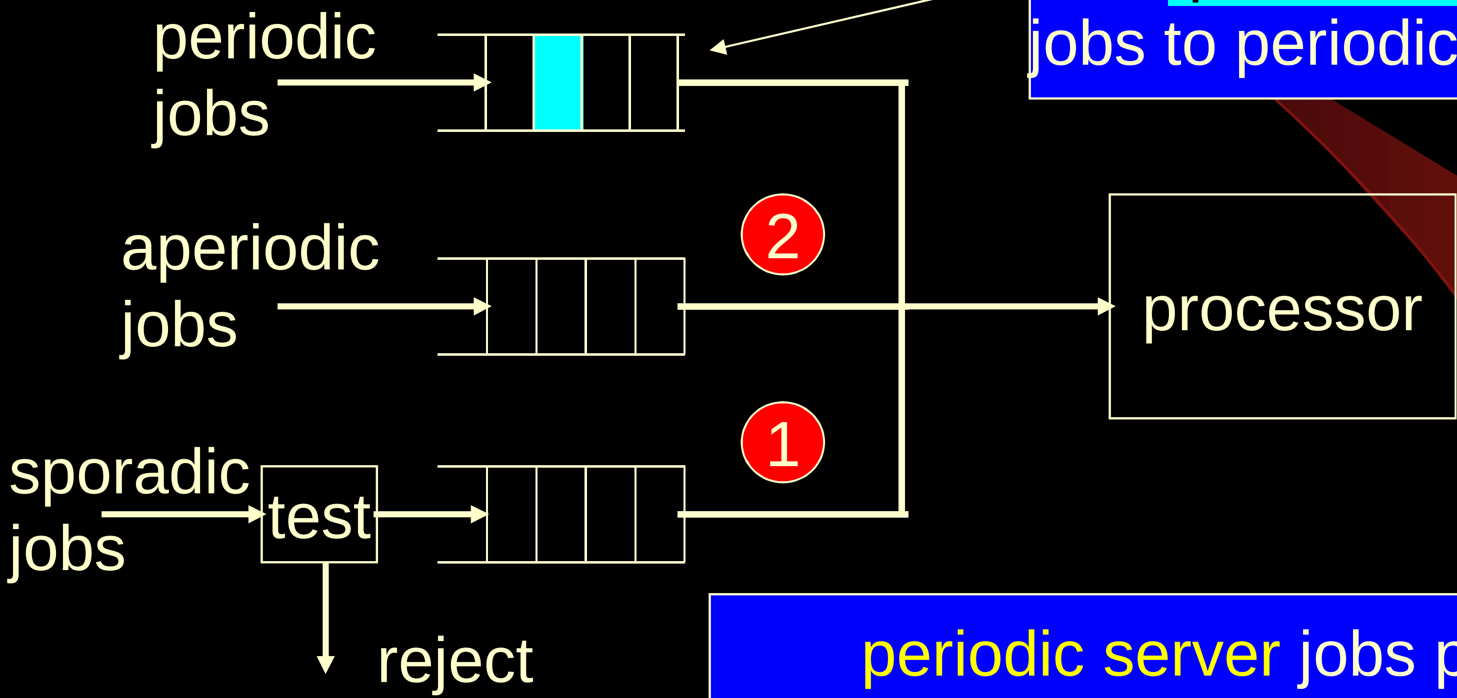
1. must have enough slack for the new job
2. accepting won't make previously accepted jobs late

→ won't influence jobs with earlier deadlines!

→ each job with a later deadline must have at least $e_{s,i}$ slack at t

Priority Queues in Kernel

add 'periodic server' jobs to periodic queue



periodic server jobs process:
(1) sporadic and then (2) aperiodic jobs